Adriano Constantino de Sousa Strumbudakis
Licenciatura em Engenharia Informática

FairSky: Gestão Confiável e Otimizada de Dados em Múltiplas Nuvens de Armazenamento na Internet

Dissertação para obtenção do Grau de Mestre em Engenharia Informática

Orientador: Prof. Doutor Henrique João Domingos, Professor Auxiliar, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa – Departamento de Informática

Júri:

Presidente:	Prof. Doutor João Baptista da Silva Araújo Junior
Arguentes:	Prof. Doutor Rui Miguel Soares Silva
Vogais:	Prof. Doutor Henrique João Lopes Domingos

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Fevereiro, 2014

FairSky

Agradecimentos

Nesta dissertação de mestrado, apesar de um processo individual, á qual qualquer investigador está destinado, há contributos que não podem nem devem deixar de ser realçados. Por essa razão desejo os meus sinceros agradecimentos:

Ao Professor Doutor Henrique João Domingos, meu orientador, pelo imenso apoio, acompanhamento do trabalho e as valiosas contribuições para o trabalho, assim como pelas críticas e sugestões, ao longo do ano que esta tese foi desenvolvida.

À minha namorada Andreia Filipa Silva Marques, pelas inúmeras trocas de impressões, correções e comentários ao trabalho realizado como também na elaboração deste relatório, especialmente em termos de linguagem. Pela grande compreensão, incentivo, paciência, e acima de tudo o grande apoio inestimável ao longo deste ano.

À minha família por estar sempre do meu lado e pela sua grande compreensão, amor, carinho e incentivo recebido ao longo da investigação.

Ao meu amigo Guilherme Silva Pedrosa, pelo apoio e incentivo dado ao longo do desenvolvimento desta dissertação.

Resumo

A utilização simultânea de múltiplas nuvens heterogéneas para armazenamento de dados remotos, disponíveis em diferentes soluções de diversos provedores desses serviços na Internet, pode ser encarada numa abordagem de organização de um repositório de dados em nuvem de nuvens de armazenamento de dados. A conceção de um sistema baseado nesta visão possui dimensões de investigação em aberto, particularmente em contextos de utilização confiável e otimizada de dados críticos, onde a aproximação tem particular interesse.

O objetivo desta dissertação visa propor, implementar e avaliar uma solução de confiabilidade, que conjuga a manutenção de propriedades de segurança e de tolerância a falhas ou intrusões nas nuvens de armazenamento utilizadas, com critérios de utilização otimizada de dados distribuídos e replicados nas múltiplas nuvens utilizadas. A dissertação propõe o sistema **FairSky**, um sistema que concretiza um repositório de dados geograficamente replicados em diferentes nuvens de armazenamento de dados, tirando partido da diversidade e heterogeneidade das mesmas e atendendo à otimização ou balanceamento de diferentes critérios: resiliência face a falhas ou intrusões bizantinas que ocorram de forma independente em cada nuvem; manutenção de requisitos de confidencialidade, integridade e autenticidade dos dados armazenados; otimização de critérios de latência de acesso aos dados; e otimização do custo inerente à utilização de diferentes nuvens de diferentes provedores. O sistema **FairSky** evita a gestão da complexidade do balanceamento dos anteriores critérios ao nível das aplicações. Apresenta-se como uma solução genérica, sob a forma de uma solução *middleware* que visa suportar diferentes aplicações, otimizando de forma transparente os anteriores critérios, tendo por base as características das diferentes nuvens que podem ser utilizadas, informação de caracterização dinâmica da operação dessas nuvens e o perfil de operação e carga (*workload profile*) das aplicações. A gestão dos dados (através do sistema *middleware*) faz-se assim com transparência e com garantias controladas de disponibilidade, tolerância a falhas, segurança, otimização de acesso e otimização de custos financeiros, tendo em conta os requisitos próprios das aplicações.

Palavras-chave: *Computação e armazenamento de dados em nuvem; replicação geográfica de dados em múltiplos centros de dados em nuvem; Gestão confiável de dados; Otimização de utilização de múltiplas nuvens de armazenamento; perfis de operação de nuvens de armazenamento de dados; perfis de aplicações para gestão, distribuição e acesso aos dados.*

Abstract

With the recent increased popularity of cloud storage services, companies, organizations and individuals have shown a lot of interest in adopting such products. A problem arises when the user has to deal with sensitive and critical data that requires high-availability and fault tolerant traits and combined with the low-cost cloud solutions.

Cloud storage providers offer a good variety of solutions for clients, with a very attractive pricing model, which often offers guarantees of scalability, availability ubiquitous access and often aided with business continuity. Many pricing models follow the pay-as-you-go model that allows clients to tailor their costs even furthers. And by reducing the client's operating costs becomes more and more appealing.

However, when dealing with critical and sensitive data, storing it in third party cloud services may become vulnerable to an outside attacker or even to the third party itself. The guarantees given by those cloud providers are limited and often clients have no assurance that they are really being carried out. Such information may be, for example, medical records, financial records, client records or any data that may compromise the client's information or even business. On the other hand adopting multiple clouds may be the immediate solution regarding availability may prove to be not very cost-wise since the different pricing models may lead to an excessive bill.

In this thesis we propose a solution to deal with both issues, a fair optimal use of the clouds and a tool to store data in a confidential a private way that will also guarantee fault tolerance with data replication. All data will be indexed and distributed throw-out the clouds available. There is a profiling tool that will be monitoring the transactions between the client and the system as to provide a long term adjustment of the balance optimizing the costs.

Keywords: *Cloud storage and computing; Safe and reliable management of replicated data in cloud storage; Reliable management of data across multiple storage clouds; Optimized use of multiple storage clouds; Cloud computing and data storage profiling; Application profiling for data repository access.*

Conteúdo

Lista de Tabelas

Lista de Figuras

1

Introdução

1.1. Contexto e Motivação

Há cada vez mais empresas a investirem parte do seu capital em tecnologias e sistemas de informação, sendo direcionado para esse sector a gestão e armazenamento de dados de modo seguro e fiável, de forma a que os mesmos sejam acessíveis em tempo útil com critérios adequados de disponibilidade, o que envolve preocupações de salvaguarda ou replicação para eventual recuperação face a falhas, devido a acidentes ou resultante de intrusões que os coloquem em causa. Embora os custos de armazenamento para a própria empresa tenham vindo a reduzir-se ao longo dos últimos anos, a utilização de infraestruturas e sistemas computacionais em centros de dados privados, bem como eventuais custos de licenciamento de *software* numa base de custo total de propriedade e manutenção, consome uma grande percentagem do investimento ao qual ainda acresce o custo dos recursos humanos dedicados. Várias empresas de provimento de serviços de armazenamento remoto de dados ou de suporte de aplicações, disponibilizando as suas infraestruturas computacionais e soluções de *software* como serviços disponibilizados em nuvens de computação ou armazenamento de dados surgiram para atender à anterior questão. As nuvens de serviços de armazenamento de dados oferecem características apelativas e interessantes para empresas e utilizadores que lidam com um volume cada vez maior de dados e que não dispõem da infraestrutura necessária ou que optam por pagar para um serviço de baixo custo de armazenamento [1, 2, 3].

Nos últimos anos, as soluções disponíveis como nuvens de computação ou de armazenamento de dados (ou soluções diferenciadas de *computing clouds* ou *storage clouds)*[1] tornaram-se muito populares, sendo hoje cada vez mais adotadas por empresas, organizações e particulares. Tendo em vista possíveis soluções diferenciadas de nuvens de armazenamento operadas como serviços na Internet, encontramos hoje uma explosão de soluções, algumas com vocação mais abrangente e com cobertura de centros de dados à escala planetária e suporte para elevada escalabilidade [1]–[3], outras para mercados de utilização mais localizados, que podem ser nacionais ou regionais [4], [5], algumas com maior enfoque para suporte de utilização individual e suporte mais ou menos especializado para aplicações específicas, embora abertas a integração para outros fins [6]–[10] e outras mais vocacionadas para comunidades de utilizadores mais específicas[11]–[13]. Estas soluções, oferecidas por diferentes provedores desses serviços na Internet, fornecem aos utilizadores uma grande diversidade de serviços e modelos de custo, podendo ir de soluções de *outsourcing* de armazenamento de dados, *outsourcing* de infraestruturas computacionais como serviços (*IaaS* ou *Infrastructure-as-a-Service),* onde o cliente tem acesso a um conjunto de processadores, memória e armazenamento, ou acesso a plataformas de *software* como serviços (*Platform-as-a-Service),* onde o cliente tem acesso a um ambiente de execução e acesso a uma aplicação específica (*Software-as-a-Service)* geralmente acedida através de um *browser* mas também a partir de aplicações especializadas. Todos estes serviços seguem modelos de pagamento dinâmicos por uso (*pay-per-use*) ou estáticos, por patamares de utilização, com alguns fornecedores a disponibilizarem pacotes limitados gratuitos.

1.2. Utilização de múltiplas nuvens de armazenamento

A utilização combinada e em simultâneo, de diferentes soluções, vista como uma nuvem de nuvens, tem sido uma aproximação explorada por diferentes aproximações da investigação, o que permite beneficiar de diferentes vantagens[14]–[16]. Esta utilização é porém problemática tendo-se em vista conjugar e otimizar diferentes critérios, nomeadamente: em relação a prever as melhores condições de latência e acesso aos dados; as melhores condições de disponibilidade dos dados com base na sua replicação por diferentes centros de dados de diferentes provedores; ou como forma de controlar os custos necessários de utilizar diferentes soluções. Neste tipo de soluções a otimização dos custos é, por sua vez, um problema que se pode revelar bastante complexo dada a diferença entre os tarifários existentes, sendo um critério difícil de conjugar

[1] Ao longo do documento que está escrito em língua portuguesa, utiliza-se frequentemente o termo cloud na terminologia convencional da área em vez de nuvem, tendo em vista a pragmática da utilização e interpretação conceptual do termo no contexto da dissertação.

com requisitos de frequência de acesso aos dados para leitura ou para escrita ou com requisitos de replicação, tendo como motivação diferentes requisitos de consistência.

Para lidar com estes problemas, a gestão da anterior complexidade poderia ser endereçada por cada aplicação, em função do seu perfil de requisitos, como um problema a resolver pelos programadores de cada uma das aplicações e parametrizações específicas caso a caso. Esta aproximação não parece uma abordagem interessante na busca da resolução dos problemas anteriores a partir de uma solução mais genérica, capaz de proceder à seleção dinâmica e ótima de diferentes nuvens de armazenamento e diferentes centros de dados dos provedores dessas nuvens, numa visão única de um serviço transparente de armazenamento de dados utilizando múltiplas nuvens e centros de dados, apresentando-se a solução de uma forma uniforme reutilizável por diferentes aplicações[14].

A maioria dos principais fornecedores de nuvens de armazenamento na Internet[1]–[3], [6], [9] oferece, na prática, um nível elevado de disponibilidade e fiabilidade do serviço, muitas vezes superiores às condições de manutenção dos dados em infraestruturas próprias dos utilizadores, com vantagens acrescidas de acesso ubíquo e otimizado, através de centros de dados espalhados pelo mundo. Estes serviços suportam operações de escrita (PUT) ou leitura (GET), com transparência da gestão dos recursos de armazenamento distribuídos na sua infraestrutura e de forma a que o acesso aos dados se faça nas condições ótimas beneficiando da "proximidade" em latência entre algum dos seus centros de dados e os clientes. No entanto, as soluções de possível replicação dos dados entre diferentes centros de dados é deixada para as aplicações. Por exemplo, não obstante existirem oito centros de dados à escala mundial representando a infraestrutura global do provedor Amazon S3[1], cada aplicação terá que replicar por si própria os dados nesses diversos centros de dados, com base em operações de leitura e escrita dos dados em cada centro de dados, separadamente. Sem qualquer controlo de monitorização de métricas de latência, as aplicações deverão replicar os dados por todos os centros de dados, se quiserem garantir cegamente que os dados estarão sempre nas melhores condições de latência pata efeitos de acesso. Por outro lado, a replicação indiscriminada de objetos em diferentes centros de dados incorre em custos que podem tornar-se significativos, devendo mais uma vez a gestão do balanceamento entre critérios de custo e de minimização de latência ser resolvido pelas aplicações. Como foi observado por alguns autores em trabalhos de investigação recente[17]–[19] a automatização do aprovisionamento de configurações de otimização de custos em função de caracterizações de carga e perfil de requisitos das aplicações revela-se uma direção de trabalho muito relevante, quer na exploração de diferentes centros de dados de um mesmo provedor, quer na adoção mais genérica de diferentes centros de dados e diferentes provedores[14].

Por outro lado, a utilização combinada de diferentes centros de dados e diferentes provedores, beneficiando da sua diversidade e heterogeneidade, pode também revelar-se benéfica como estratégia de segurança e tolerância a falhas. Estas dimensões revelam-se problemáticas quando os dados são críticos, como acontece com dados privados, informação financeira ou dados de registos médicos, entre outros. Assim as aplicações desenvolvidas para lidar com esses dados exigem requisitos de segurança, fiabilidade e disponibilidade, numa solução conjugada de confiabilidade, que tem que ser obtida a partir das múltiplas nuvens, não necessariamente confiáveis [15], [16]. De outro modo, as empresas ou os particulares vão recear que os seus dados sejam expostos a ataques de intrusão contra cada um dos provedores bem como sejam objeto de acesso e utilização incorretos por parte de pessoal do próprio fornecedor.

Finalmente existe a necessidade de evitar situações em que os fornecedores possam tirar partido de dependências dos utilizadores de modo a exercerem práticas para contrariar a vontade de mudança de provedor pelos utilizadores, dificultar o acesso aos dados ou aumentar o custo dos serviços, práticas identificadas em alguns autores como práticas do tipo *vendor lock-in*[15].

Em resumo, revela-se assim interessante que numa solução de nuvem de nuvens possam ser estabelecidos critérios de balanceamento e otimização de condições de latência e de custos, combinados com estratégias de replicação dos dados, combinando esses critérios com propriedades de disponibilidade e tolerância a falhas, ao mesmo tempo que se devem poder suportar condições de segurança, nomeadamente de privacidade, autenticidade, integridade e controlo de acesso aos dados. Numa solução deste tipo, poderão então ser disponibilizadas aos utilizadores critérios de auditoria e controlo, quando as nuvens de armazenamento utilizadas não podem fazer parte da base de confiança da solução.

1.3. Problema

Não existem soluções que resolvam todos os problemas de segurança, fiabilidade disponibilidade e ótimo uso do espaço e recursos de armazenamento em *cloud*. Para garantir privacidade e confidencialidade usam-se técnicas de criptografia bem conhecidas e estabelecidas, enquanto que para fiabilidade e tolerância a falhas recorre-se a técnicas de replicação com um certo nível de tolerância a falhas fornecendo garantias de recuperabilidade dos dados.

Por outro lado as soluções abordadas tratam as *clouds* de forma uniforme referente aos custos, suportando embora alguma heterogeneidade das mesmas, causando possíveis dificuldades de aplicação das mesmas num ambiente limitado ao custo. Várias empresas fornecem diferentes modelos de serviços com diferentes tipos de transações a serem taxadas tornando difícil um uso ótimo de um conjunto de *clouds* para garantir replicação fiável. Por outro lado essas

clouds podem ou não permitir configuração de geolocalização dos dados, fornecendo acessos mais rápidos, torna-se

A utilização simultânea de múltiplas nuvens heterogéneas para armazenamento de dados remotos, em diferentes soluções de provedores desses serviços na Internet, pode ser encarada numa abordagem de organização de nuvem de nuvens de armazenamento de dados, com vantagens que podem ser exploradas para efeitos de confiabilidade, na conjugação de diferentes critérios nas suas diversas dimensões: segurança por diversidade, privacidade dos dados, disponibilidade bem como fiabilidade com tolerância a falhas ou intrusões ao nível de cada um dos provedores.

A conceção de um sistema confiável baseado nesta visão de uma nuvem de nuvens, sendo inicialmente uma aproximação sedutora, possui no entanto desafios e dimensões de investigação em aberto. Estes desafios são particularmente complexos em contextos de utilização confiável de dados críticos, na qual a aproximação tem particular interesse.

É sabido que soluções de armazenamento em nuvem apresentam hoje diferentes opções de contratualização para critérios de utilização, na oferta de diferentes provedores. As características de qualidade de serviço por parte dos operadores bem como os critérios particulares de aferição do desempenho operacional são ainda pouco conhecidos e estudados. Na linha que se pretende investigar na presente dissertação, torna-se no entanto interessante utilizar as soluções disponíveis atendendo à conjugação de diferentes fatores-chave associados a critérios de confiabilidade, análise e otimização da operação e controlo de custos.

Do ponto de vista de confiabilidade é necessário assegurar condições de privacidade, confidencialidade e integridade dos dados mantidos em infraestruturas computacionais remotas, sem controlo direto dos utilizadores. As anteriores propriedades de segurança devem ser garantidas em conjugação com mecanismos de controlo de disponibilidade dos dados, tolerância a falhas ou ataques por intrusões que ocorram ao nível dos provedores. Tal exige que todas essas propriedades se estabeleçam, o mais possível, com controlo e independência dos utilizadores em relação aos provedores, o que pode ser conseguido numa solução de nuvem de nuvens.

Por outro lado, a adequabilidade das soluções exige que se possa dispor de critérios de análise dinâmica de qualidade de serviço ou de métricas de operação e custo das soluções, o que requer a caracterização dos requisitos específicos das aplicações e seu mapeamento nos critérios de confiabilidade e desempenho das diversas nuvens de armazenamento, aspeto ainda muito pouco explorado na investigação atual.

1.4. Objetivo da Dissertação

O objetivo da dissertação é propor, conceber, materializar e testar uma solução baseada numa arquitetura de sistema *middleware* que permite a utilização conjugada de diversas nuvens de armazenamento de dados disponíveis como serviços Internet. O sistema concretiza uma visão de utilização transparente de múltiplas nuvens de armazenamento de diferentes provedores Internet, como uma nuvem de nuvens heterogéneas. A diversidade das soluções é desde logo um critério primário para alicerçar condições de segurança, disponibilidade e fiabilidade da solução. As diversas nuvens são utilizadas como repositórios flexíveis de dados, não necessariamente confiáveis, para suporte de aplicações. Nesta flexibilidade pretende-se explorar garantias de confiabilidade, combinando critérios de segurança e fiabilidade. Como condições de segurança o sistema endereça propriedades de autenticidade, confidencialidade, privacidade e integridade dos dados armazenados, utilizando técnicas criptográficas que nunca permitirão expor os dados ao nível da infraestrutura computacional dos provedores das nuvens de armazenamento utilizadas. Como condições de fiabilidade o sistema endereça propriedades de disponibilidade permanente dos dados com base em critérios e replicação e fiabilidade do acesso para recuperação dos dados, com tolerância a falhas ou intrusões que ocorram de forma independente ao nível das nuvens utilizadas.

No sentido do anterior objetivo a dissertação apresenta o sistema **FairSky**, um sistema que concretiza assim um repositório de dados geograficamente replicados em diferentes nuvens de armazenamento de dados e respetivos centros de dados, tirando partido da diversidade e heterogeneidade das mesmas e atendendo à otimização ou balanceamento dos critérios acima enumerados, de forma combinada com otimização de critérios de latência do acesso aos dados e custos da utilização de diferentes nuvens de armazenamento.

O sistema **FairSky** evita a gestão da complexidade do balanceamento dos anteriores critérios ao nível das aplicações. Apresenta-se como uma solução genérica, com uma arquitetura *middleware*, permitindo disponibilizar a sua base de serviços para possível suporte de diferentes aplicações. A otimização dos anteriores critérios faz-se com base (i) nas características das diferentes nuvens que podem ser utilizadas e informação de caracterização dinâmica da operação dessas nuvens em termos de métricas de latência ou de custos, (ii) a partir de configurações dos serviços de *middleware*, e (iii) tendo em conta o perfil de operação das aplicações[20], métricas de *workload* e resiliência para tolerância a falhas. A gestão dos dados faz-se assim de forma transparente para as aplicações, com garantias controladas de disponibilidade, tolerância a falhas, segurança, otimização de acesso e otimização de custos financeiros.

1.5. Principais Contribuições

Tendo por base o anterior objetivo, as principais contribuições da presente dissertação decorrem da criação de um sistema de armazenamento de dados seguro e ótimo para múltiplas *clouds* que forneça as seguintes propriedades de confiabilidade e usabilidade da solução:

- **Confidencialidade**: De modo a que o fornecedor de uma dada cloud, ou atacante externo nunca possa ter acesso aos dados, sendo os mesmos cifrados recorrendo a métodos de criptografia simétrica.[21], [22]
- **Integridade**: De modo a garantir que os dados não são alterados por fatores externos, recorrendo a funções de síntese e integridade. [23]
- **Suporte para múltiplas clouds de armazenamento**: recorrendo a módulos responsáveis por interagir com API específica de cada cloud fornecendo assim uma camada de abstração.[21], [24]
- **Fiabilidade e tolerância a intrusões**: com recurso a mecanismos de replicação de dados recorrendo a múltiplas clouds podendo ser acedidos e verificados a modos de serem reconstruídos facilmente.[23], [24]
- **Disponibilidade**: face a eventuais falhas de uma ou mais clouds ou por ações maliciosas ou legais, desde que o numero de clouds usadas permitam a reconstrução dos dados em questão.[24]
- ***Profiling***: permitir a análise de dados e consoante configurações distribuir os dados de modo a minimizar os custos para o utilizador[25] bem como impacto de latência pela utilização conjugada de múltiplas nuvens de armazenamento, permitindo que os tempos de acessos ao sistema sejam mantidos de forma a atenderem requisitos de perfil de aplicações.

Tendo em conta as anteriores propriedades, as contribuições da dissertação consistem:

- Na proposta de um sistema baseado numa arquitetura de middleware, capaz de ser concretizado e utilizado transparentemente por diferentes aplicações, como repositório virtual de objetos, tendo por base uma arquitetura de nuvem de nuvens

1.6. Organização do documento

Seguido este capítulo introdutório, no Capítulo 2, é feito um levantamento do estado da arte em relação aos sistemas de armazenamento de *cloud*, que oferecem ou não garantias de integridade,

redundância ou confidencialidade dos mesmos. São abordados inicialmente as questões e problemáticas relacionadas com segurança e integridade, sendo assim apresentados algumas soluções interessantes para a abordagem das questões como também uma breve descrição das falhas bizantinas. São descritas um pequeno conjunto de fornecedores de armazenamento *cloud* com uma breve descrição do modelo de serviços. Por fim apresenta-se a outra problemática importante para esta dissertação referente a questão de *profiling* de dados como também umas aproximações interessantes para resolver esta questão.

No capítulo 3, é apresentada a estrutura e visão arquitetural de uma solução *middleware* que responde as questões e objetivos apresentados nesta tese, sendo seguido pelo capítulo 4 que apresenta a versão implementada como também os detalhes da mesma com as questões importantes. Sendo a implementação é apresentada, testada e validada face a um conjunto de testes e medições apresentadas no capítulo 5. Finalmente no capítulo 6 efectua-se um resumo e conclusão da dissertação e apresenta-se o trabalho futuro.

2

Estado da arte

Neste capítulo apresentam-se e discutem-se várias referências importantes de trabalho relacionado com visão o desenvolvimento de um *middleware* que possa endereçar com os objetivos apresentados no capítulo anterior. Na secção 2 são apresentadas as noções essenciais que são importantes para este trabalho. Em 2.2 são apresentados algumas implementações práticas com vista em objetivos de privacidade, integridade e tolerância a falhas, alguns dando mais prioridade numas áreas que outros, como também uma breve descrição do conceito de tolerância a falhas. A secção 2.3 contém um número de fornecedores de armazenamento em *cloud* com foco nos modelos de venda e funcionalidades disponíveis. Em 2.4 aborda-se a problemática de análise e monitorização dos conteúdos da aplicação como também recolha de métricas das diversas *clouds* a modos alcançar os objetivos da dissertação. Finalmente em 2.5 é apresentado o trabalho relacionado em resumo, sendo também analisadas questões de segurança e *profiling* das soluções referidas ao longo deste capítulo.

2.1. Segurança dos Dados

Tendo em conta o aumento drástico do uso de serviços *clouds* na última década, sendo o mesmo considerado um fator muito importante para o desenvolvimento e sucesso das empresas [26], tendo assim como consequência os fornecedores destes, aumentarem a gama de conteúdo oferecido. Os sistemas em *cloud* levantam um grande número de questões relativamente a segurança dos dados, a integridade dos mesmos e até a disponibilidade sob diversas formas de ataques [27]. Em seguida iremos apresentar alguns deles.

2.1.1. Privacidade e Confidencialidade

Os dados armazenados nas diversas *clouds* estão sujeitos a ataques internos provenientes do próprio provedor, mesmo não sendo intencional pela empresa em si, mas por um administrador com acessos[27], sendo que não existe transparência para o cliente sobre quem tem acesso aos dados por parte da empresa fornecedora. Casos de ataques efetuados por terceiros como o exemplo de ataques com máquinas virtuais no serviço Amazon EC2 recorrendo técnicas de virtualização onde a máquina maliciosa pode via execução no mesmo servidor físico que uma máquina de um outro cliente/vítima, ter acesso as informações guardadas em memória[28] dado que diversas *clouds* alojam vários clientes e a infraestrutura necessita de ter acesso aos dados desencriptados para os poder processar e efetuar operações sobre eles. Alternativamente podem considerar o fornecedor do serviço em si, duvidoso a modos de querer ocultar informação do mesmo. Uma outra questão importante, apesar das garantias que os diversos fornecedores podem fornecer, a falta de transparência e a impossibilidade de demonstrar perante um cliente essas garantias obriga a adoção de modelos de confidencialidade e privacidade.

Existem várias maneiras e soluções para cifrar os dados dependendo do nível de segurança que queremos garantir recorrendo a cifras simétricas, cifras homomórficas entre outros. Em [29] são apresentados 2 grandes esquemas para fornecer privacidade e confidencialidade perante os diversos tipos de ataques apresentados anteriormente. A primeira abordagem passa por particionar a aplicação em camadas com cada camada a ser executada numa *cloud* distinta. Deste modo permite a arquitetura da aplicação desenvolvida manter numa *cloud* os dados e em outra a parte de operações e a lógica da aplicação em si, assim se os dados estiverem protegidos fornece uma camada de ofuscação para cada uma das *clouds* sobre que dados são mantidos Figura 2.1. Embora esta primeira abordagem seja interessante poderá revelar informação a terceiros recorrendo aos acessos aos dados via monitorização por parte do atacante das leituras e escritas efetuadas, sem mencionar que os dados para poderem ser processados pela camada lógica estarão possivelmente desprotegidos.

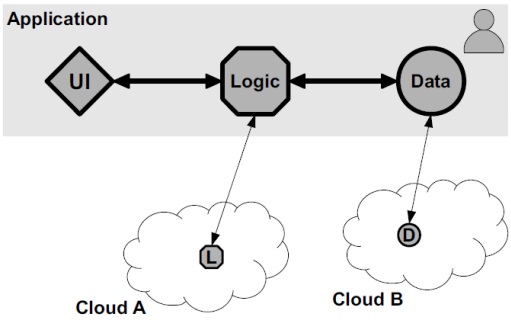

Figura 2.1: Separação da aplicação dos dados

Assim foi proposta a separação da aplicação em camadas similarmente a questão anterior mas a aplicação também será fragmentada por mais que uma *cloud* sendo que todos os dados não serão processados por um único fornecedor, como poderá ser visto na Figura 2.2. Esta abordagem poderá ser feita recorrendo a duas aproximações consoante a fragmentação efetuada. Ofuscando a separação, onde os dados são distribuídos de modo a nenhuma única *cloud* ter informação completa sendo limitadas a uma pequena fração dos mesmos, enquanto que os dados finais estarão confidenciais na parte do cliente. Um exemplo proposto é substituir os identificadores dos dados de acordo que a não revelar nenhuma meta informação durante as operações executadas entre as diversas *clouds*. Finalmente é proposto uma abordagem de computação com múltiplos participantes, onde cada participante efetua calcula partes dos dados e usando técnicas de partilhas de segredos distribui essas partes para *clouds* diferentes. As *clouds* em seguida irão efetuar cálculos sobre essas partes. Por fim, essas partes serão enviadas de volta ao cliente que assim poderá reconstruir o resultado. Deste modo é garantido a confidencialidade dos dados introduzidos a exceção da situação onde todas as *clouds* comuniquem entre si para deliberadamente revelar os dados.

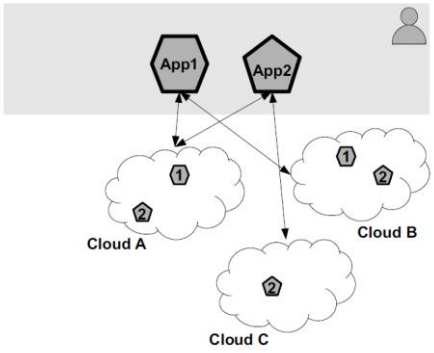

Figura 2.2 : Fragmentação da aplicação

2.1.2. Integridade e Disponibilidade

Uma outra problemática introduzida é referente negação de acessos aos dados devido a ataques maliciosos, questões legais ou mesmo falhas do próprio fornecimento, adicionalmente existe a questão dos dados poderem corromper-se ou corromper os mesmos, dando ao cliente informação falsa. Um destes exemplos de negação de acesso aos dados foi em Abril de 2009 [30]. Onde o estado americano apreendeu servidores de um centro de dados, baseado em suspeitas que atividades relacionadas em crimes informáticos. Nesses mesmos centros de dados encontravam-se também alojados dados de outros clientes. Assim vários clientes viram negados o acesso aos seus dados armazenados nessas e noutras clouds que sofreram destinos semelhantes, vários desses negócios sofreram bastante até mesmo fechando. Semelhante a questões legais ou de fornecedores podemos ter ataques de negação de serviço a fornecedores (*Denial of Service*), que po-

derá impossibilitar de forma semelhante acesso aos dados, ou até políticas de *vendor lock-in* onde o fornecedor intencionalmente nega o acesso. Isto cria a necessidade de ter os dados recuperáveis a partir de outro local, seja do mesmo fornecedor ou de outro, visto serem em muitos casos importantes para a viabilidade de empresas relacionadas com tecnologias de informação.

Uma outra questão semelhante é a questão de integridade de dados, onde os mesmos podem sofrer alterações intencionais por parte de atacantes maliciosos ou por questões alheias. Desta forma existe a necessidade de garantir que os dados estão corretos, recorrendo a técnicas de assinaturas nomeadamente funções de *hash*, as quais servem para informar que um dado conteúdo não sofreu alteração, técnicas de recuperabilidade, como será explicado mais a frente, falhas bizantinas, onde garante-se a tolerância a falhas permitindo a reconstrução dos dados quando um conjunto de servidores/serviços falha. Adicionalmente a dados corrompidos podemos expandir a corrupção também a questões de computação caso a aplicação em questão dependa da *cloud* para isso, havendo servidores que poderão reportar informação falsa semelhante a dados corrompidos, para este caso poderá adotar-se similarmente uma solução de tolerância a falhas. Em [29] propõe-se um sistema de redundância onde na questão de computação a aplicação recolhe os cálculos de n serviços havendo f falhas, assim poderá assumir que para um numero recolhido de $n - f > f$ respostas pode-se assumir que a maioria dos serviços cloud são honestos permitindo concluir na resposta correta baseando na comparação da maioria dos resultados obtidos.

2.2. Gestão confiável de dados

Hoje em dia é cada vez maior número de empresas a oferecer serviços de armazenamento de dados, fornecimento de recursos ou de uma combinação de ambos, sobre as formas de modelos de serviço:

- **IaaS**: Infraestrutura como serviço, o modelo mais básico de fornecimento, onde o cliente usa diretamente os recursos da cloud permitindo controlo direto sobre o sistema operativo.
- **PaaS**: Plataforma como serviço, modelo onde o cliente tem acesso a um conjunto de ferramentas e *frameworks* para poder executar as suas operações, não tem controlo sobre o sistema operativo sendo esta uma camada superior de abstração.
- **SaaS**: *Software* como serviço, considera-se a camada superior de abstração, delegando a manutenção exigida para o fornecedor do serviço. Habitualmente uso deste tipo de modelo foca-se mais para serviços Web.

Questão que coloca vários problemas no que toca problemáticas e confianças em terceiros que gerem essas *clouds*. Existem problemáticas relativas com a disponibilidade dos recursos, como questões de pontos de falha, negação de acessos devido a casos de excesso de condições não estipuladas nos contratos, corrupção de dados e de integridade dos dados e das operações. Problemáticas relacionadas com níveis de segurança como ataques as *clouds* diretamente e finalmente questões de controlo nomeadamente, espionagem por parte do fornecedor do serviço ou até acesso aos dados em si, ou mesmo casos de *vendor lock-in*[31]. Questões que várias empresas colocam antes de exportar dados levando a questões de não uso ótimo das clouds em si e por vezes adotando soluções de *clouds* híbridas, *clouds* compostas por fornecedores públicos e *clouds* privadas.

Em seguida apresentam-se algumas soluções estudadas no âmbito deste trabalho que propõem resolver a maioria das questões anteriormente apresentadas.

2.2.1. iDataGuard

Este sistema proposto apresenta um componente de *middleware* a ser executado no cliente ou num intermediário com confiança[21]. As aplicações exportam o seu sistema de ficheiros para este *middleware* que por sua vez recorre a um conjunto de *clouds* para armazenar os dados, sendo o mesmo responsável por guardar e obter os dados das *clouds*, enquanto que as mesmas não percebem da existência do mesmo visto para elas não passar de ser uma aplicação. Para alcançar este tipo de comunicação os autores optaram por usar adaptadores para cada uma das *clouds* criando assim uma camada de abstração, a fins de combater a heterogeneidade entre os vários fornecedores, dado que as operações agora suportadas pelo sistema limitam-se a inserções, atualizações, leituras e remoções de conteúdo.

Todos os dados do sistema encontram-se cifrados por chaves geradas a partir de uma palavra mestre introduzida pelo utilizador e em função das iterações do algoritmo também, de modo a que não seja possível por atacantes recriar a chave. Os objetos guardados nas diversas *clouds* são compostos por vários atributos: o conteúdo do elemento guardado, o nome do elemento, os meta-dados e um identificador gerado pelo módulo CryptInd[32], para permitir indexação do ficheiro. O iDataGuard para além de ter como foco a confidencialidade dos conteúdos, opta adicionalmente por esconder a estrutura do sistema de ficheiros implementado, recorrendo a especificação da estrutura dentro dos meta-dados que vão cifrados como também os outros atributos a exceção do identificador. Querendo assim prevenir ataques provenientes de terceiros que possam obter informação baseada na estrutura Este sistema também visa preservar integridade dos diversos objetos, assim recorre a *HMACs* os quais são calculados e guardados por par-

te dos servidores. Estes mesmos são compostos por um identificador de versão do objeto de modo a garantir que o cliente recebe sempre o mais atualizado evitando conflitos.

Uma das questões importantes neste tipo de sistema é a pesquisa sobre os conteúdos cifrados nos fornecedores. Para dar face a este problema os autores usam uma segunda aproximação, particionando o índice de cada objeto por um número elevado de vezes, propondo assim a versão CryptInd++[32], permitindo pesquisas por padrões. Recorre-se a *q-gram*, formando um novo índice com o conjunto de palavras que compõem o dado *q-gram*. O exemplo apresentado no modelo relativo a esta abordagem apresenta a pesquisa de um subconjunto de caracteres de uma palavra procurando assim a palavra em questão, ao qual existe uma palavra a qual fornece o identificador do conteúdo completo permitindo assim obter o conteúdo baseado numa pesquisa parcial, ao contrario do modelo inicial que necessitava de uma lista de identificadores para permitir indexar os conteúdos.

No âmbito deste trabalho esta solução proposta pelo iDataGuard em conjunto com um sistema de tolerância a falhas bizantinas cria uma solução apelativa, no enquanto existem algumas limitações relacionadas com a escalabilidade do sistema e puderam haver casos onde a solução não seja viável, nomeadamente quando um cliente executa um pedido de indexação completa da estrutura. A necessidade de uso de índices auxiliares poderá não ser o ideal visto inserir rondas de comunicação adicionais, que neste trabalho poderão causar aumento do tráfego exigido ao fornecedor.

2.2.2. Silverline

A solução proposta pelo conjunto de ferramentas fornecidas no Silverline, embora não focada como a anterior em sistemas de ficheiro, mas em dados sobre uma base de dados. Para fornecerem confidencialidade os dados são encriptados ao máximo possível, tendo em conta que haverá sempre dados que não podem ser cifrados dado serem utilizados por aplicações de terceiros, nomeadamente o exemplo apresentado sobre utilizadores dos quais os dados estão cifrados a exceção das datas de nascimento a fins de permitir cálculos pela aplicação de *cloud* sobre a idade média dos mesmos.

Por outro lado para não haver impacto no desempenha do sistema, e permitir acessos a grupos de dados a grupos de utilizadores de modo a não expor a um único toda a informação caso seja desnecessário, o sistema analisa o programa sobre o qual é executado, observando as operações que o mesmo efetua e os dados que os clientes operam individualmente definindo os grupos de acesso necessários. Deste modo clientes só podem visualizar conteúdo para o qual lhes foi autorizado acesso, recorrendo unicamente as chaves que cada cliente tem, guardadas

localmente. Assim poderá haver clientes com mais que uma chave consoante o grupo de permissões nos quais se inserem.

Este sistema proposto apresenta certas vantagens para refrescamento de chaves visto não necessitar de cifrar de novo os objetos guardados, situação que iria necessitar rondas de comunicação e computação adicional, causando impacto indesejável no sistema global, alternativamente o sistema opta por outras alternativas que a curto prazo provam ser mais eficazes. Nomeadamente quando um utilizador é removido em vez de cifrar os dados de novo, o sistema emite uma nova chave para os dados armazenados, atualizando os clientes restantes do grupo com a nova chave. Que por outro lado ao manter este tipo de informação poderá levar a longo prazo criação de certas inconsistências e existência de *overhead* desnecessário.

No âmbito deste trabalho esta aproximação poderá aplicar-se a fornecedores de *SaaS* nomeadamente de bases de dados, poderá também ser interessante aplicar a metodologia de processamento e gestão de chaves no caso o trabalho que ira ser desenvolvido utilizasse mais que um utilizador em simultâneo. Na questão de processar e analisar a aplicação *cloud* a ser utilizada por este sistema é um fator interessante na questão de *profiling* de operações efetuadas.

2.2.3. HAIL

O sistema HAIL[23], propõe um sistema de ferramentas para permitir o uso de múltiplos serviços de armazenamento em simultâneo, possibilitando ao cliente consultar se um ficheiro armazenado esta intacto e é recuperável. Este sistema baseia-se em dois aproximações básicas, provas de recuperação (*proof of recovery* - POR) e provas de posse (*proof of possesion* – PDP). Recorrendo a estes o cliente pode facilmente pedir a um servidor que lhe prove que um dado ficheiro existe e não está corrompido, para este efeito o cliente pode simplesmente limitar-se a consultar uma fração do ficheiro, mas por outro lado exige computação do lado do servidor, fator que nem sempre é possível em qualquer serviço de *cloud.* Este sistema oferece garantias de recuperabilidade de ficheiros visto o modelo de adversário considerado é de um atacante móvel, mais concretamente de um atacante que pode corromper ou controlar um subconjunto de servidores em simultâneo mas nunca todos os servidores num dado momento, permitindo as várias outras réplicas reconstruírem os ficheiros. Enquanto que os servidores não comunicam diretamente existindo assim uma comunicação limitada a cliente-servidor, essa não contem um alto acréscimo na comunicação visto permitir computação do lado do servidor incluindo otimizações para minimizar rondas excessivas.

Relativamente a deteção de servidores defeituosos, o cliente fica encarregado de comunicar com os outros servidores cujos fragmentos dos ficheiros corrompidos estão intactos a modos de atualizar o servidor em falha. Para este efeito o HAIL recorre a três elementos importantes.

Códigos de dispersão, os quais distribuem os dados de forma robusta nos diversos servidores, mais concretamente usam código de correção de erros com proteção de integridade recorrente a *MACs*. Código de servidor, concretamente os blocos de ficheiros encontram-se codificados com código de correção, protegendo contra corrupção de baixo nível dos conteúdos caso falhem os testes de integridade. E por fim código de agregação, onde o servidor comprime as respostas para os clientes, nomeadamente em combinar os diversos *MACs* num único, sendo assim mais eficiente verificar vários blocos em simultâneo, sendo computado em tempo real a modos de não ser necessário de ocupar espaço adicional em disco. Como pode ser visto na Figura 2.3 onde um ficheiro é repartido, distribuído por diversos servidores como também blocos adicionais a modos de permitir reconstruir e recuperar os ficheiros durante falhas.

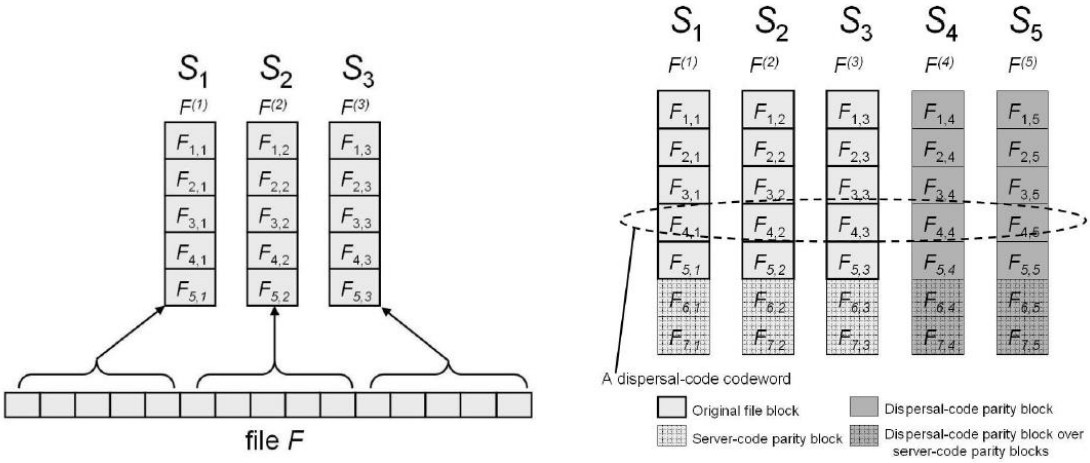

Figura 2.3 : Fragmentação do sistema HAIL

Em relação a este trabalho proposto, o sistema HAIL apresenta uma boa solução distribuir um conjunto de ficheiros por várias *clouds,* recorrendo a técnicas interessantes para tolerar e recuperar falhas, minimizando em simultâneo a sobrecarga no cliente. Por outro lado este sistema adota a necessidade de existência de computação por parte dos fornecedores das diversas *cloud* a usar, facto que poderá impedir de ser usado em sistemas limitados a armazenamento. Um outro aspeto importante nesta implementação é o modelo de adversário e o modo de verificação de falhas, que embora seja adequado considerando que existe maneira de recuperar erros, a metodologia de consultar fragmentos de ficheiros permite ao atacante corromper fragmentos ficheiros. Situações onde um cliente consulta fragmentos consistentes de ficheiros que no seu todo estão corrompidos tendo falsa informação sobre o todo de um ficheiro.

2.2.4. DepSky

Para oferecer garantiras de disponibilidade, integridade e opcionalmente de confidencialidade sobre múltiplas *clouds* apresenta-se o sistema DepSky[24]. Foi apresentado um protótipo deste sistema usando tecnologias Java, criptografia simétrica, *erasure codes* e a biblioteca de *Java Secret Sharing* (JSS) para partilha de chaves, tendo também, similarmente aos sistemas apresentados, uma aproximação cliente-servidor, recorrendo a comunicação HTTPS REST.

São apresentadas duas vertentes do sistema DepSky-A, para garantir integridade e disponibilidade dos dados, e a versão DepSky-CA onde para além das garantias do sistema anterior permite confidencialidade dos dados armazenados. Mais concretamente na questão de disponibilidade dos dados o DepSky-A réplica os mesmos por mais que um fornecedor de serviços *cloud* necessitando por mais espaço globalmente. No que toca a falhas bizantinas o sistema garante capacidade de recuperação para um conjunto de $\mathbf{3}\boldsymbol{f}\mathbf{+}\boldsymbol{1}$ para $\boldsymbol{f}$ *clouds* em falha ou com dados corrompidos. Para garantir a integridade dos dados, os mesmos são assinados digitalmente sendo guardadas as assinaturas nas próprias *clouds*. No sistema DepSky-CA o qual oferece confidencialidade o sistema permite que os dados estejam distribuídos pelos diversos servidores a modos de ser necessário obter $\boldsymbol{f}\mathbf{+}\boldsymbol{1}$ partes para reconstruir o segredo, sendo assim para um atacante com $\boldsymbol{f}$ partes impossível de recriar os dados originais, partilhando assim um segredo S recorrendo a biblioteca JSS, existindo uma entidade própria para distribuir esse segredo.

Dos diversos testes apresentados na proposta, observa-se que em comparação com uso direto das *clouds*, o sistema apresenta um grau de impacto mínimo sendo o mesmo inferior a 2% do tempo. Dadas algumas particularidades de algumas *clouds*, observou-se que ficheiros grandes (≥1Mb) apresentam maiores latências que ficheiros respetivamente menores (≤100Kb), enquanto que o acesso a ficheiros seja melhor em função do tamanho dos ficheiros dada a replicação dos mesmos sendo possível obter os fragmentos por mais que um fornecedor ao mesmo tempo, a questão introduzida de partilha de ficheiros causa impacto considerável ao sistema global.

No âmbito deste trabalho, esta solução enquadra-se com alguns objetivos propostos. Sendo que apresenta uma tolerância a falhas bizantinas aceitável com um número mínimo de réplicas necessárias, capacidade de confidencialidade e integridade dos dados armazenados, enquanto que não é abordado em profundidade a questão de indexação dos dados por parte do cliente.

2.2.5. Falhas Bizantinas

Os sistemas anteriormente apresentados visam resolver uma problemática de falhas, denominadas de falhas bizantinas, que foram primeira vez apresentadas em 1982[33]. Neste artigo a pro-

blemática foi exposta sobre um exemplo de cenário de guerra onde a comunicação entre comandantes não é fiável, nem mesmo garantir que os comandantes são leais, não havendo indivíduos maliciosos. Este modelo propõe teoricamente que para garantir dados coerentes necessitam-se de ***3 f + 1*** participantes para ***f*** participantes maliciosos. Foram apresentadas outras soluções para esta problemática.

2.2.5.1 O modelo PBFT[34] que foca-se em ambientes assíncronos pressupondo comunicação entre os servidores recorrendo a uma máquina de estados determinista replicada. O cliente efetua um pedido e desencadeia-se um protocolo de confirmação de integridade constituído por 3 fases: *pre-prepare*, onde é difundida a operação para todas as réplicas, *prepare* onde as réplicas respondem o sucesso da operação e a operação de *commit* onde as réplicas sincronizam-se e prosseguem a resposta ao utilizador.

2.2.5.2 O sistema Query/Update[35] recorre a quóruns de repostas para garantir tolerância a falhas. Ao contrário do modelo teórico apresentado este modelo usa 5 f + 1 para f falhas. Baseia-se em comparação de historiais de operações como também métodos HMAC para a garantia de integridade. Durante a deteção de falhas o sistema permite atualizações de históricos ou mesmo recorrendo a acesso exclusivo ao sistema para recuperar as réplicas em divergência de estados.

2.2.5.3 O modelo Hybrid Quorum Replication[36] usa uma combinação de máquinas de estados e quóruns. Recorre a um número de 3 f + 1 réplicas igual ao modelo teórico. Quando não há período de contenção usa-se os quóruns enquanto que durante períodos de contenção usa-se a máquina de estados para evitar conflitos de acessos.

2.2.5.4 O sistema Zyzzyva[37] que baseia-se no anterior PBFT mas recorre a especulação para reduzir os custos. Assim se as respostas estão coerentes e consistentes assume-se que o sistema está consistente, caso contrário prossegue-se para a fase de sincronização.

2.2.5.5 Por fim o UpRight Cluster Services [38] surge apresentando uma biblioteca que usa filas de espera para sincronizar os quóruns de respostas. A comunicação entre as réplicas é efetuada recorrendo a verificação de *digests* para garantir a consistência do estado e das operações efetuadas.

2.3. Fornecedores e soluções cloud

Nesta secção são apresentados vários serviços de armazenamento em *clouds*. Todos eles apresentam diferenças nas políticas de segurança como também diferentes operações e limitações.

Importantes para este trabalho são também os planos e tarifários que cada um apresenta como também os diferentes conteúdos de cada plano. Mais concretamente em seguida descrevem-se alguns desses fornecedores com informação importante relativa para este trabalho.

- **Amazon S3**[2]: Permite aceder aos dados recorrendo a *buckets*, os quais encontram-se replicados e distribuídos por várias máquinas, permite também especificar localização dos mesmos para otimizar as latências de operações. Proporciona um conjunto de operações sobre os mesmos recorrendo a uma API SOAP ou REST.
- **Box**[3]: Oferecem garantias de replicação e tolerância a falhas, toda a comunicação pode ser cifrada e os dados são também cifrados. Propõem plano de dados gratuito para um único utilizador com um limite de 5Gb de armazenamento enquanto que um ficheiro tem como limite de 250Mb de tamanho, havendo possibilidade de contratar maiores limites. Por outro lado a privacidade especificada menciona que poderão existir terceiros com acessos aos dados impondo necessidade de garantir confidencialidade dos dados por parte do utilizador. Na conta gratuita existe um limite de 10GB de tráfego mensal, enquanto que os ficheiros têm um prazo de validade de 120 dias. Fornece também uma API proprietária.
- **Google Drive**[4]: Permite acesso remoto a um conjunto de tipos de ficheiros para visualizar ou alterar recorrendo a interface web fornecida, nomeadamente Google Docs. Permite similarmente um serviço gratuito com 5Gb de espaço inicial, com possibilidade de expansão até 16Tb. Pode-se operar sobre os conteúdos criando ou editando dados recorrendo a uma API fornecida.
- **Google Cloud Storage**[5]: Similarmente a Amazon S3, opera sobre *buckets*, sendo possível permitir também localização dos mesmos. Fornece acesso direto aos conteúdos, como também uma API REST. Não existe plano gratuito como na Box e Google Drive começando com um tarifário taxando o espaço usado, o tráfego efetuado como também as operações sobre os *buckets*.
- **Microsoft SkyDrive**[6]: Similarmente ao sistema Google Drive, o serviço SkyDrive fornece inicialmente 7 Gb com possível expansão até 125Gb, enquanto que o tamanho dos ficheiros esta limitado a 2 Gb. Os conteúdos armazenados são eliminados após 270 dias de inatividade da conta associada que poderá causar questões de armazenamento a longo prazo. Também permite acesso aos dados com as operações essências

[2] http://aws.amazon.com/s3/ (acedido em 29/1/2013)
[3] http://www.box.com/ (acedido em 30/1/2013)
[4] https://drive.google.com/ (acedido em 30/1/2013)
[5] https://developers.google.com/storage/ (acedido em 30/1/2013)
[6] http://skydrive.com/ (acedido em 30/1/2013)

de escrita, leitura atualização e remoção via uma API REST também. Permite manter versões dos conteúdos armazenados.

- **Microsoft Azure Storage**[7]: Este serviço armazena os dados de modo replicado. Fornece três tipos de armazenamento: *Blobs, Tables* e *Queues* representando métodos diferentes de armazenar os dados. Um *blob* representa um ficheiro identificado por um nome único sobre os quais podemos efetuar operações de Get, Put, Copy e Delete. As tabelas servem para mapear dados, nomeadamente uma tabela é composta por linhas e colunas. Finalmente as *queues* representam filas de mensagens as quais permitem operações de filas sobre as mesmas como o nome indica. Os planos de custos baseiam-se em espaço ocupado, largura de banda usada, havendo também a opção de usar base de dados SQL.
- **Rackspace Cloud Files**[8]: Os dados armazenados neste fornecedor são replicados por diversos servidores, em diferentes localizações geográficas. Permite armazenamento recorrendo a contentores os quais podem ser privados ou públicos consoante o acesso que o cliente pretende ter. A comunicação é feita recorrendo a uma API REST com segurança SSL. Não existe modelo inicial gratuito havendo taxação para o espaço usado e o tráfego realizado com origem no sistema, não havendo custos de *upload.*
- **Ubuntu One**[9]: Um serviço similar o SkyDrive. Oferece um plano gratuito de 5Gb de espaço, havendo opção de pagar por mais. Fornece uma API REST para aceder aos conteúdos armazenados, não havendo custos para o tráfego utilizado. Os dados não se encontram cifrados por parte do fornecedor enquanto que o mesmo recorre ao serviço da Amazon S3 para o espaço fornecido, havendo imediatamente a questão de o controlo dos dados pertencer a terceiros.
- **Dropbox**[10]: Similarmente a SkyDrive e Ubuntu One, este fornecedor permite a existência de planos gratuitos como também planos pagos consoante o espaço usado. O plano inicial inclui 2Gb de espaço. Permite adicionalmente partilhar conteúdo caso o cliente o especifique, sobre este conteúdo é imposto um limite de 20Gb por ficheiro por mês enquanto que num plano pago o limite aumenta para 200Gb, fatores a considerar no desenvolvimento do trabalho proposto. Os ficheiros estão alojados, similarmente ao Ubuntu One, na Amazon S3 embora estejam cifrados também. O serviço fornece uma API própria para efetuar operações sobre os conteúdos.

[7] http://www.windowsazure.com/ (acedido em 30/1/2013)
[8] http://www.rackspace.co.uk/cloud-files/ (acedido em 30/1/2013)
[9] http://one.ubuntu.com/ (acedido a 30/1/2013)
[10] https://www.dropbox.com/ (acedido a 30/1/2013)

2.4. *Profiling*

2.4.1. Com visão para o desempenho das *clouds*

Hoje em dia com o aumento do número de empresas de fornecimento de serviços *cloud*, seja armazenamento, computação, bases de dados ou mesmo combinação dos mesmos, aumenta a necessidade de poder permitir comparar os serviços fornecidos a modos de permitir aos potenciais clientes escolher um ou mais fornecedores de acordo com as especificações necessárias. Em [39] são apresentados diversos serviços *online* de empresas que fornecem informação sobre métricas importantes para o possível utilizador. Duas destas ferramentas apresentadas são CloudHarmony[13] e CLOUDSLEUTH[12].

O primeiro serviço monitoriza a disponibilidade dos serviços de *clouds* desde 2009 tendo uma grande gama de fornecedores observados desde *IaaS* soluções como Rackspace e Amazon Web Services até *PaaS* como Microsoft Azure e Google App Engine. Inicialmente o cliente é disponibilizado 5 *tokens* com a possibilidade de adquirir mais para uso. O utilizador em seguida tem que decidir na medição que deseja consultar como também o conjunto de fornecedores a visualizar sendo os resultados demonstrados numa tabela havendo gráficos para visualização fácil. As métricas oferecias tem foco fornecedores de soluções de computação em *cloud*, nomeadamente métricas relativas com os processadores sendo cálculos para único processador ou multiprocessador, métricas de I/O recorrendo várias técnicas de obtenção de valores tanto para disco ou memoria, mais concretamente bonnie++[11], IOzone[12], STREAM, entre outros. Por outro lado não existe informação se os dados disponibilizados são selecionados baseados em médias ou representam máximos e mínimos por cada serviço.

O segundo serviço apresentado, CLOUDSLEUTH[13], fornece informação relativamente as latências e disponibilidade dos centros de dados dos fornecedores sendo os mesmos mapeados num mapa global. Sendo este tipo de serviço mais interessante face aos objetivos propostos. Por outro lado não existem resultados específicos que possam ser obtidos, de modo similar ao CloudHarmony[14], desde modo não é possível obter uma noção do I/O de rede.

2.4.2. Com visão para os ficheiros

Em [25] é proposto um mecanismo de decisão sobre uso de ficheiros a modos de permitir os administradores de servidores libertarem espaço para utilização, sendo que os ficheiros com

[11] http://www.coker.com.au/bonnie++/ (acedido a 30/1/2013)
[12] http://www.iozone.org/ (acedido a 30/1/2013)
[13] https://cloudsleuth.com/ (acedido a 30/1/2013)
[14] http://cloudharmony.com/ (acedido a 30/1/2013)

pouca ou nenhuma utilização serem armazenados em fitas magnéticas ou outro tipo de dispositivo de armazenamento a longo prazo. Foi proposta a criação de uma ferramenta de monitorização baseada numa versão modificada da ferramenta GNU de procura *find*, com fins de não ter que criar módulos de *kernel* próprios. Esta ferramenta reúne informação relativa ao tamanho de cada ficheiro, a data de último acesso, a data de última modificação e o proprietário do ficheiro. Como visto em [40] é observado que em 80% dos ficheiros criados num sistema operativo tem tempo de vida menos de três minutos, levando a não considerar estes para o sistema de crescimento a longo prazo.

Inicialmente são abordados quarto algoritmos, um baseado em filas FIFO, outro em menos recente uso (*least-recently used - LRU),* outro em tamanhos e finalmente um recorrendo a combinação de tamanho e tempo (ST). As filas FIFO são simples de implementar sendo o seu algoritmo muito básico e fácil. Deste modo os ficheiros são arquivados consoante o tempo de permanência no sistema. Problemas são levantados em questões que um ficheiro com idade, por exemplo um ano, que esteja a ser frequentemente utilizado pode ser arquivado mais cedo de um outro com nenhum acesso devido ao facto de o segundo ter sido criado mais recentemente. O segundo apresenta desempenho bom para sistemas de memória, enquanto que a maior desvantagem apresenta ser o facto de o algoritmo ignorar o tempo necessário de transferir esses dados para o arquivo. Uma variante do segundo algoritmo é a terceira proposta analisada, enquanto que o LRU recorre a tempos de uso, o de tamanhos recorre ao tamanho do ficheiro, apresentando as desvantagens semelhantes as filas FIFO. O último algoritmo analisado recorre a uma combinação entre o tamanho do ficheiro e o algoritmo LRU, sendo no estudo proposto a formula $Space * Time^{1.4}$ sendo assim migrados os ficheiros que apresentam maior coeficiente desta fórmula.

Apresenta-se desta forma uma solução baseada em valores de envelhecimento dos ficheiros, recorrendo adicionalmente a toda a informação adicional observada pela ferramenta de monitorização, tendo em consideração também o uso dos ficheiros no passado, fator que não é levado em conta pelos quatro algoritmos estudados. Deste modo ficheiros com uso frequente tornam-se candidatos fracos para o processo de arquivação enquanto que ficheiros com pouco uso têm fator aumentado. O algoritmo proposto tem em consideração o historial de uso do ficheiro, assim ficheiros que não tem acessos com alguma regularidade são maiores candidatos a serem arquivados em comparação com ficheiros que são acedidos com alguma regularidade, mesmo tendo o mesmo número de acessos. Os estudos realizados apresentam uma otimização de 2 a 20 vezes melhor do algoritmo proposto face ao ST, tendo em conta também o número de ficheiros que não foram processados pelo ST.

2.5. Conclusão

2.5.1. Resumo

Neste capítulo foram abordados e analisados várias problemáticas referente aos dados dos utilizadores que colocam em serviços de terceiros, nomeadamente questões de privacidade referente a quem pode ter acesso aos dados, questões de confidencialidade referente a não revelar nenhuma informação sobre os dados mantidos tanto seja a terceiros ou até mesmo ao fornecedor do armazenamento, enquanto também foram expostas questões de integridade e disponibilidade dos dados, tanto face a ataques por elementos maliciosos ou mesmo por falta de tolerância a corrupção de dados pelo fornecedor ou situações onde os dados que colocam-se nos mesmos não estão disponíveis devido a políticas ou falhas.

Uma problemática referente as garantias previamente abordadas é a existência de tolerância a falhas, saber recuperar das mesmas tanto em questões de integridade como disponibilidade. Foi abordado a questão de falhas bizantinas, um exemplo onde com um dado conjunto de elementos não existe nenhuma garantia relativamente quais desses elementos estão em falha necessitando assim uma metodologia para determinar a resposta correta recorrendo a um conjunto de elementos os quais consideramos corretos. Dentro desta problemática foram apresentadas vários sistemas para tolerar falhas deste tipo, cada um com o seu conjunto de objetivos e abordagem da mesma questão, desde o modelo teórico até a soluções com implementação proposta. As soluções estudadas recorrem a máquinas de estados como o PBFT[34] e o Zyzzyva[37], enquanto outros recorrem a quóruns como o Query/Update[35] e o UpRight[38]. Alguns destes sistemas inserem rondas de comunicação adicional para assegurar a integridade dos dados, que poderá causar inconveniências nas questões de tempo de resposta ou o fluxo de dado.

Foram apresentados um conjunto de soluções práticas com visão as problemáticas apresentadas. Começando pelo iDataGuard[21], onde é apresentado como um *middleware* que fornece uma camada de abstração para as diversas clouds apresentando um único sistema ao cliente, atingindo a confidencialidade encriptando os dados e meta dados dos objetos com chaves derivadas de uma chave mestra. O sistema Silverline[22] analisa os dados da aplicação e o acesso aos mesmos a modos de criar grupos de utilizadores com acessos limitados aos dados que lhes são atribuídos, assim esta abordagem tem como prioridade a privacidade dos dados num ambiente multiutilizador tratando os dados como base de dados. Considerando o uso de múltiplas *clouds*, o HAIL[23] recorre a distribuição e fragmentação dos dados com uso de provas de recuperação, tendo como objetivo a simples e eficaz comunicação tendo assim garantidas as questões de integridade e disponibilidade dos recursos. Finalmente o sistema DepSky[24] simi-

larmente ao HAIL tem objetivos principais a disponibilidade de integridade dos dados mas recorre a implementação de um sistema em torno da abordagem teórica das falhas bizantinas.

Na secção 2.3 foram enumerados vários fornecedores de serviços *clouds*. Regra geral, são serviços baseados em repositórios de dados, muitas vezes replicados internamente fornecendo tolerância a falhas numa infraestrutura distribuída. Para garantirem baixas latências alguns fornecedores optam por replicar os dados geograficamente. Estas fornecem acesso aos dados do utilizador de forma ubíqua para serem facilmente acedidas via interfaces WEB, REST ou casos específicos recorrendo a API próprias. Cada fornecedor opta por tratar os dados do cliente de formas diferentes, alguns permitem certa transparência permitindo ao utilizador interagir com o seu espaço sob a forma de repositórios (*buckets*) ou acesso tradicional recorrendo a API REST. Importante nesta secção é salientar as diferenças entre os modelos de pagamentos oferecidos ao cliente havendo fornecedores que taxam ligações diferentes (*upload* e *download),* enquanto que outros oferecem planos gratuitos para particulares impondo limites de tráfego de modo diferente que em planos pagos.

Por fim na secção 2.4, é apresentada uma questão bastante relevante para os objetivos do trabalho. São apresentados duas abordagens, uma com foco nas diferentes *clouds,* onde é importante destacar o facto de poder comparar as várias métricas entre si, nomeadamente questões de tempos de acesso, tempos de escrita, latências da rede entre outras. A outra abordagem endereça metodologias de *profiling* referente a ficheiros no cliente para um ambiente distribuído, havendo algumas soluções e algoritmos propostos para escolher os ficheiros com menos uso, a serem arquivados para minimizar o espaço ocupado nos servidores ou escolha de *clouds* com base no desempenho das mesmas. Questões abordadas recorrendo a fornecedores externos destes serviços, que abordam questões de extração de métricas recorrendo a computação na *cloud,* não serão abordadas neste trabalho.

2.5.2. Analise critica

Face aos objetivos propostos desta dissertação e os vários sistemas estudados, não existe um que englobe as funcionalidades pretendidas. Com o enorme aumento de oferta de serviços de *cloud* com diferentes modelos de tarifários, mais e mais empresas estão inclinadas em adotar soluções deste tipo para os seus negócios. Assim torna-se importante para as mesmas haver garantias de segurança, confidencialidade, disponibilidade, privacidade e também uma forma de poder otimizar os recursos disponíveis recorrendo a múltiplas *clouds* sem que haja excessos de gastos ou que seja possível disponibilizar os conteúdos em tempos reduzidos.

Referente as questões de segurança, integridade e disponibilidade, esta dissertação poderá seguir algumas aproximações apresentadas na realização dos objetivos. Nomeadamente o siste-

ma DepSky apresenta uma solução embora parcial interessante para as questões de integridade e disponibilidade, considerando adicionalmente a vertente DepSky-CA[24] para atender as questões de confidencialidade e privacidade. Visto que as *clouds* em vista não possuem capacidade de computação, não podemos ter situações onde os dados são baseados em comunicações entre servidores. Adicionalmente na questão de comunicação, visto a mesma ser taxada em certas ocasiões, a abordagem proposta de eleger uma réplica como réplica principal para confirmar se os dados estão corretos demonstra ser a mais adequada visto que abordagens de aguardar um numero elevado de réplicas poderá ser dispendioso. Casos de recuperação de erros que deverão ser levadas a cabo pelo cliente. Adicionalmente existe a problemática de permitir o sistema poder interagir com diversas *clouds* heterogéneas, uma problemática que o sistema DepSky aborda com sucesso.

Para replicação de dados o sistema HAIL[23] apresenta uma forma de distribuição e fragmentação dos dados, interessante. Permite fragmentar um bloco por diversas *clouds* minimizando o tráfego necessário para consultar a integridade das mesmas embora a garantia oferecida não seja a melhor devido a um atacante poder corromper dados que não são consultados de modo a levar a crer ao cliente que os dados tão consistentes. Ao replicaremos os blocos podemos garantir tolerância a falhas de modo a não sobrecarregar o espaço disponível, fator importante no âmbito deste trabalho. Sendo assim a garantia de integridade alcançada recorrendo ao mesmo sistema HAIL mas com possível acréscimo de algoritmos de *hash* para cada bloco em si, sendo facilmente verificável caso os dados do bloco estejam corrompidos.

Na questão de *profiling* de uma aplicação, o sistema Silverline[22] esta mais próximo visto a efetuar operações de análise e recolha de informação para poder permitir um sistema com garantias de privacidade de dados para múltiplos utilizadores. Na questão dos objetivos propostos o sistema será semelhante mas terá objetivo o sistema de ficheiros que deverá ser analisado a modos de reduzir os espaço necessário e as invocações dispendiosas caso os ficheiros em questão estejam alojados em fornecedores com alto custo de comunicação para o cliente. Alternativamente o sistema permitira escolher e operar sempre sobre as clouds cujas métricas de latência favoreçam repostas em tempos reduzidos. O sistema abordado na secção 2.4.2 apresenta uma ferramenta importante na questão de análise de ficheiros permitindo prever qual a probabilidade de um dado ficheiro ser usado. Esta aproximação enquadra-se nos objetivos deste trabalho, visto as clouds não permitirem extrair dados usadas em tempo real sendo que seria necessário que as mesmas permitissem computação. Todos os cálculos para esse efeito deverão ser efetuados pelo lado do cliente, opcionalmente poderão ser consultados para o cálculo dessas métricas dados a partir de ferramentas online como vistas em 2.4.1.

3

Arquitetura

Tendo em vista o capítulo anterior, onde foram abordadas as problemáticas e contribuições com visão a modelar um sistema que cumpra os objetivos apresentados, neste capítulo iremos apresentar e elaborar o modelo de arquitetura que se enquadre na problemática como também a arquitetura mais especifica que será apresentada e usada nos capítulos seguintes de implementação e testes. Inicialmente será abordada a visão geral do sistema seguida do modelo de arquitetura o qual terá três vertentes de possíveis funcionamentos.

3.1. Apresentação geral da solução proposta

Com os objetivos da dissertação em vista e com as contribuições relevantes para esta tese podemos destacar que a visão geral do sistema proposto será composto por três grandes componentes. Numa ponta os diversos e heterogéneos provedores de armazenamento na *cloud.* Na outra ponta os clientes finais deste sistema que poderão ser um ou mais. Sendo que entre os provedores e os clientes reside o *middleware* que ira tratar de ligar as operações efetuadas pelos clientes as *clouds* ligadas ao sistema, como se pode observar na Figura 3.1, essas operações terão que ser realizadas de modos a respeitar e cumprir os objetivos propostos no capítulo anterior recorrendo a mecanismos internos no sistema, a modos de para além de os concretizar tornar a utilização do sistema simples, intuitiva e modelar. Todas as operações auxiliares referente a decisões intrínsecas por parte do mecanismo de *profiling* recorrendo a métricas, não são transparentes ao utilizador sendo que o utilizador ira ter acesso a um conjunto mínimo de operações de escrita e leitura de dados, sendo possível utilizar este sistema para correio eletrónico, repositório de ficheiros, gestor de base de dados entre outros.

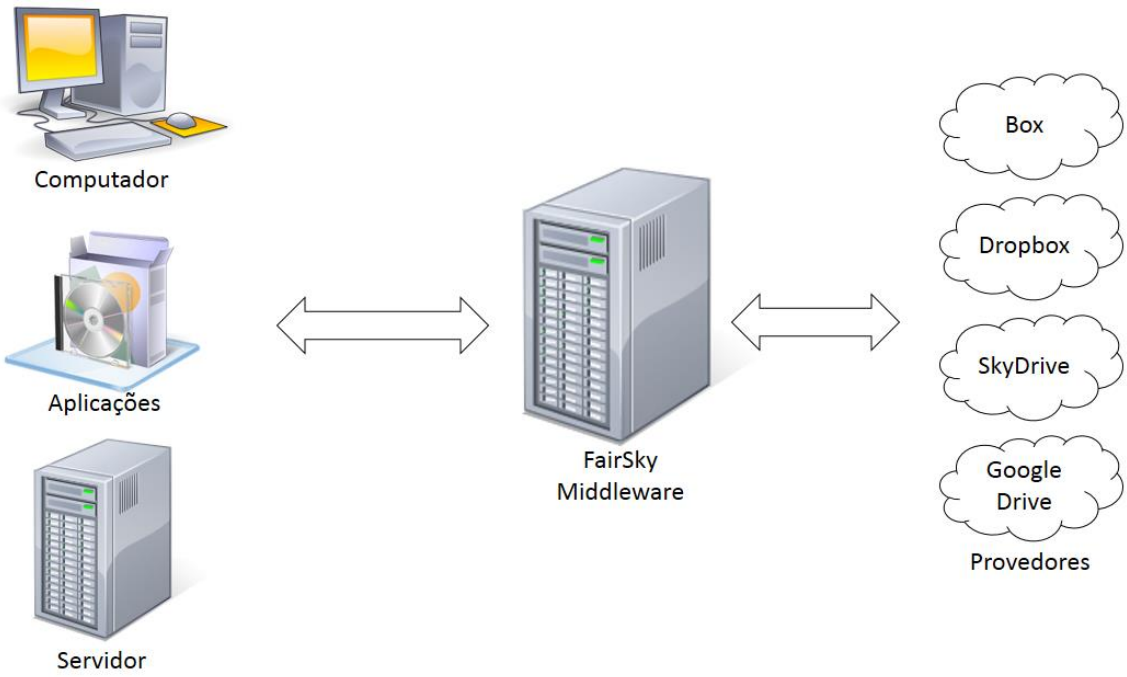

Figura 3.1: Arquitetura Geral

3.2. Modelo do Sistema

3.2.1. Modelo de Sistema

Como referido anteriormente o sistema FairSky reside entre os clientes e as aplicações, sejam elas aplicações locais ou serviços de rede. Sendo assim possível ser executado em diversos cenários, cenários esses que serão abordados mais a frente. Internamente o sistema é composto por diversos módulos.

- O módulo de segurança composto por 3 componentes interiores, a encriptação, integridade e autenticidade dos dados.
- O módulo de gestão de dados que ira coordenar as operações invocadas aos utilizadores e comunicar com os provedores para colocar ou obter conteúdo armazenado nos mesmos.
- O módulo de *profiling*, que será um oráculo, que ira disponibilizar ao gestor de dados uma lista de informação otimizada para ele efetuar as operações com melhores resultados.
- O modulo de indexação que ira manter informação crucial dos dados armazenados, permite ao sistema saber onde se encontram os dados armazenados e em que provedores.

- Os diversos conectores que permitem fornecer abstração ao sistema sobre as diversas particularidades da API dos provedores.
- O sistema de API que permite aplicações externas possam usar facilmente e com eficácia.

Estes componentes serão abordados em maior detalhe nas secções seguintes como também a forma que os dados são mantidos ao nível do middleware.

3.2.2. Modelo de Dados

O uso de múltiplas clouds requer que o middleware FairSky seja capaz de lidar com a heterogeneidade das interfaces de cada um dos provedores utilizados. Um aspeto importante do modelo de dados a usar é permitir que os mesmos sejam aceites por cada um dos provedores utilizados, sendo que este modelo permite ignorar estas particularidades nas apresentações dos algoritmos. Na Figura 3.2 podemos observar a representação do índice e o mapeamento dos objetos por parte do sistema de middleware. Como se pode observar, existe um índice de nomes de ficheiros, índice esse que aponta para um conjunto de listas de fragmentos que compõem o conteúdo do ficheiro em questão. Cada fragmento por sua vez contem informação sobre os provedores *cloud* onde esse fragmento se encontra hospedado. Tendo essa informação pode-se em seguida indexar diretamente os contentores nas clouds.

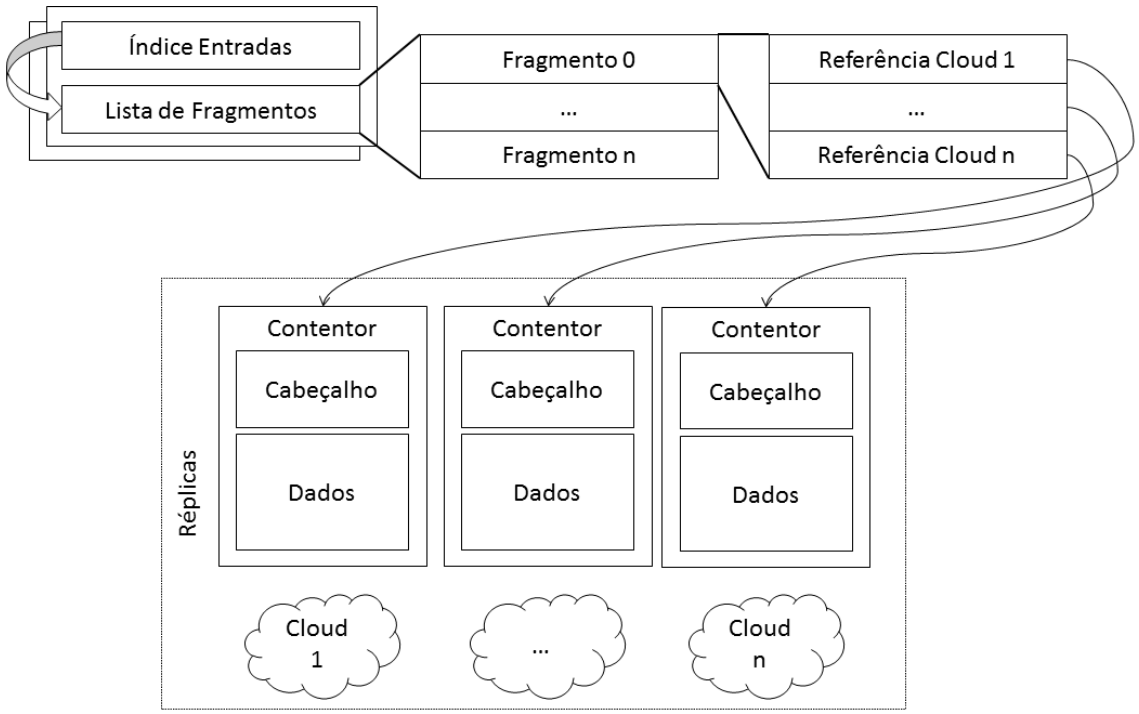

Figura 3.2: Modelo de dados do Índice

A Figura 3.3 ilustra como um objeto genérico de dados, denominado de contentor é guardado e gerido por parte do middleware, é composto pela secção de dados, que representam o conteúdo recebido e tratado do cliente e um mapa do tipo "chave-valor". Esse mapa contem toda a meta-informação relacionada com os dados e o que o contentor representa no sistema, informação essa indexada por uma chave única associada a uma sequência de dados, dados esses que poderão representar qualquer tipo de estrutura, servindo o propósito de uso da mesma, nomeadamente permitindo a cada componente do middleware utilizar e reutilizar esses campos no contentor de modo simples e com abstração.

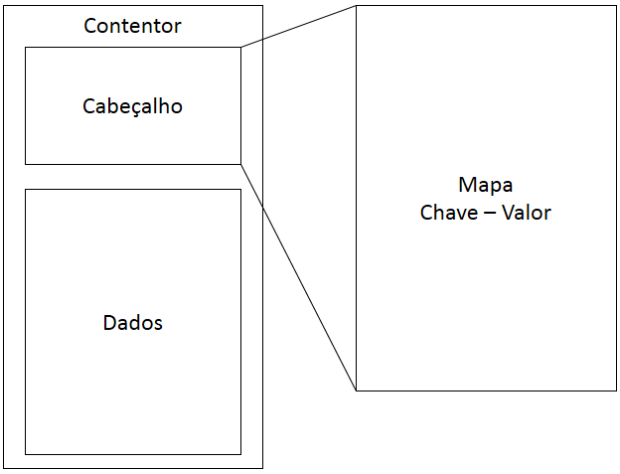

Figura 3.3: Modelo de Dados

3.2.3. Modelo de Falhas e Modelo de Adversário

3.2.3.1 Confiabilidade do Sistema

A confiabilidade do sistema alcança-se usando mecanismos de replicação e fragmentação dos conteúdos armazenados. Ao replicaremos os dados por um quórum de provedores garantimos que em qualquer momento que se um subconjunto deles não esta contactável ou os dados estão corruptos possa-se recuperar os mesmos recorrendo a replicas alojadas em outros provedores. Recuperação esta feita, recorrendo a códigos de integridade ou a comparação de réplicas entre si de forma a encontrar a réplica garantindo que o invocador da operação recebe sempre o conteúdo em questão sem que haja falhas de disponibilidade do mesmo.

Adicionalmente fragmentação de dados colocados nas clouds permite que os conteúdos sejam potencialmente dispersos por um maior número de provedores. Em casos de falhas de um provedor, os fragmentos alojados nele estão indisponíveis, logo com o aumento de fragmentos,

menor os dados a serem inacessíveis, permitindo uma recuperação mais eficaz e rápida por parte de outros provedores.

3.2.3.2 Modelo de falhas

Para este sistema consideramos um sistema de falhas ao nível de provedores, nomeadamente falhas de serviços e indisponibilidade do conteúdo, seja ele temporário ou definitivo. Deste modo um provedor em "falha", considera-se um provedor no qual o conteúdo armazenado não é possível de se aceder por parte do middleware ou por qualquer outro tipo de aplicação. Falhas deste tipo podem ter muitas origens e justificações, que no final negam acesso aos dados do utilizador.

Para este tipo de falhas é importante que o sistema tenha capacidade de aceder ao conteúdo a partir de outro provedor, permitindo assim que o utilizador tenha sempre acesso aos dados, respeitando em simultâneo, um número máximo de provedores em falha, seguindo o modelo de falhas bizantinas. Considera-se também que em momento nenhum todos os provedores encontram-se em estado de falha.

3.2.3.3 Modelo de Adversário e Serviços de Segurança

Como adversário deste sistema consideramos um atacante que tem acesso aos conteúdos armazenados nos diversos provedores, seja ele um atacante externo ou um atacante interno ao provedor. Considera-se que o dispositivo onde o FairSky executa é seguro e confiável, sendo não fazer parte deste modelo ataques ao middleware em si por parte de terceiros. Adicionalmente não são considerados para efeitos desta tese utilizadores maliciosos. Limitando assim os possíveis ataques estritamente as diversas clouds de armazenamento utilizados no middleware.

O atacante as clouds pode ter acesso aos conteúdos, modifica-los ou mesmo elimina-los. Para este efeito é necessário que o middleware seja capaz de detetar este tipo de intrusão e prevenir acessos ilícitos ao conteúdo. Recorrendo a métodos e algoritmos bem estabelecidos no nível de segurança e integridade como também autenticidade, deve-se poder concluir se um contentor colocado numa dada cloud foi corrompido, recorrendo a códigos de integridade e autenticidade e cifrar os mesmos para que não seja facilmente.

3.3. Arquitetura do Sistema e Componentes Principais

3.3.1. Arquitetura de Referência

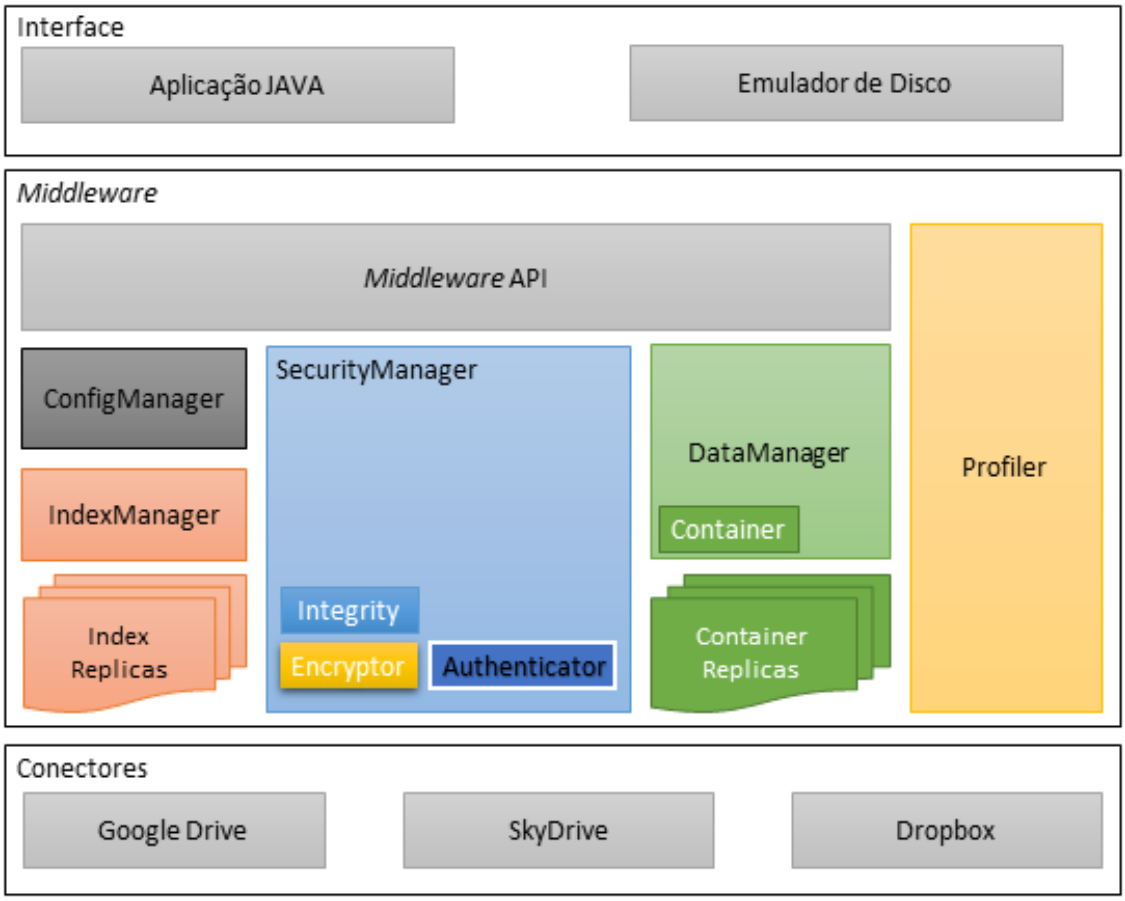

Figura 3.4: Arquitetura de Referencia

A anterior arquitetura de referência tem em vista uma concretização do sistema enquanto arquitetura middleware suportada localmente num computador de um utilizador. Desta forma, a concretização do sistema é endereçada numa arquitetura de *software* que estabelece um serviço local para suporte de execução de aplicações locais que fazem uso da funcionalidade disponibilizada sob a forma de um repositório virtual local de objetos. Este repositório virtual torna transparente às aplicações e aos seus utilizadores o acesso a diferentes nuvens de armazenamento de dados, tal como disponibilizadas por diferentes provedores destes serviços na *Internet*. Desta forma, a implementação de aplicações locais que utilizem o sistema far-se-á da mesma forma como as aplicações que implementam habitualmente a funcionalidade de um drive virtual suportado numa nuvem de armazenamento de dados (como é o caso de aplicações tais como o

GoogleDrive[15], Dropbox[16] , SkyDrive[17], Box[18] ou aplicações similares, ta como são hoje disponibilizadas para diferentes tipos de dispositivos de computação.

Tendo como base a anterior arquitetura de referência do sistema, apresenta-se de seguida a sua arquitetura de *software* e seus componentes, como refinamento e modelação do sistema com base nos seus componentes de software.

3.3.2. Arquitetura de Software

A arquitetura de software do sistema inicialmente representado na Figura 3.4, tem como base um modelo arquitetural "*Web-3-Tier",* onde as camadas superiores são camadas de interação com aplicações-cliente, enquanto que camadas inferiores tratam da interação e integração transparente de diferentes *clouds* que sejam utilizadas. A camada da lógica de middleware é composta pelos diversos componentes necessários para endereçar os requisitos e objetivos subjacentes ao modelo do sistema.

Mais concretamente, para endereçar questões de armazenamento de informação referente a objetos e sua replicação por diferentes nuvens de armazenamento de dados utiliza-se um componente de indexação (***Index Manager***). Os serviços de segurança são suportados no componente de Segurança (***Security Manager***). O suporte de transparência e heterogeneidade de uso de diversas nuvens de armazenamento é concretizado por um conjunto de conectores específicos para integração de diferentes nuvens e que concretizam instâncias concretas de uma fábrica de conectores, podendo assim suportar-se extensibilidade de adição de novos conectores, tornando transparente aos serviços do middleware a utilização específica de qualquer nuvem, bastando que a mesma suporta as operações usuais de escrita e leitura de objetos, segundo uma arquitetura de suporte do tipo "*key-value store*", como é habitual a essas soluções.

A utilização de um conector para cada uma das clouds, respeita assim uma interface abstrata de operações (como instância de definição de uma fábrica de conectores), sendo cada conector usado de forma transparente ao nível dos serviços de middleware pelo componente de repositório de dados (***Data Manager***) ao nível da camada de integração de dados da arquitetura de três níveis apresentada.

Para suporte de aplicações-cliente são disponibilizadas pela **API externa** um conjunto de operações semelhantes a operações de I/O de um sistema de ficheiros.

Para se decidir quais as melhores *clouds* a usar no momento em que uma aplicação decide realizar uma determinada operação na API externa usa-se o componente **Profiler**. Este compo-

[15] https://drive.google.com/ (acedido a 20/10/2013)
[16] https://www.dropbox.com/ (acedido a 20/10/2013)
[17] https://skydrive.live.com/ (acedido a 20/10/2013)
[18] https://www.box.com/ (acedido a 20/10/2013)

nente é usado internamente pelos serviços de middleware como um oráculo que, dado um perfil de uma aplicação e um conjunto de indicadores de monitorização das nuvens disponíveis no sistema, decide, com base num conjunto de regras declarativas, qual a nuvem ou nuvens que serão utilizadas para armazenamento e replicação dos dados. Como se explicará mais detalhadamente mais à frente, as métricas de perfil e de monitorização envolvem diferentes critérios que foram selecionados para serem suportados no componente de *profiling*, nomeadamente as seguintes:

Métricas de monitorização e controlo das nuvens utilizadas:

- Métricas de monitorização de latência
  - Latência de operações de escrita
  - Latência de operações de leitura

- Métricas de disponibilidade de armazenamento
  - Espaço disponível para o modelo de custos contratado

- Métricas de Custo Financeiro
  - Custo de operações de escrita
  - Custo de operações de leitura
  - Custo de armazenamento

Métricas estabelecidas pelo perfil de aplicações

- **Métricas de confiabilidade**
  - Número de réplicas (ou nível de resiliência) requerido
  - Nível de fragmentação das réplicas
  - Nível de priorização de acesso em leitura e recuperação de dados
  - Nível de priorização de acesso em escrita
  - Nível de priorização de acesso em pesquisa

As anteriores métricas e a sua agregação em políticas de *profiling* estabelecidas para diferentes aplicações correspondem a parâmetros fornecidos ao componente de *profiling*, podendo estas serem configuradas de forma ortogonal aos serviços de middleware. Por outro lado, as métricas de monitorização das nuvens de armazenamento registadas no middleware podem ser igualmente definidas como parâmetros do componente de *profiling*, ou podem ser obtidas por processos de instrumentação que capturam indicadores de monitorização das diversas nuvens utilizadas e submetem esses indicadores como parâmetros de entrada ao componente de *profiling*. O processamento e suporte das anteriores métricas do componente *profiling* bem como o suporte de definição de políticas e monitorização das métricas de *profiling* serão explicados em maior detalhe na próxima secção, na explicação sobre o componente **Profiler**.

Na lógica do suporte de execução dos serviços de *middleware*, o componente **DataManager** é responsável por aceitar os pedidos efetuados a API, consultar o **Profiler** e depois recor-

rendo aos conectores e ao módulo de segurança efetua a encriptação ou decifração de dados a serem colocados ou obtidos das *clouds* indicadas pelo **Profiler** anteriormente.

3.3.3. Componentes da arquitetura de *software*

Como anteriormente apresentados brevemente os macro-componentes principais que compõem a arquitetura de *software* são: a API (**Middleware API**) e seu suporte, o componente de indexação (**IndexManager**), o profiler (**Profiler**), o componente de segurança (**SecurityManager**) e o gestor de operações de dados (**DataManager**), pressupondo este, uma fábrica abstrata de conectores de nuvens e as respetivas instâncias de implementação de conectores específicos para cada nuvem de armazenamento registada e usada no sistema.

Resume-se de seguida o papel de cada um destes macro-componentes que constituem os módulos principais da arquitetura de *software* tal como foi concebida.

Middleware API: Este componente como o nome indica, tem como único objetivo fornecer ao utilizador ou as suas aplicações um conjunto de comandos semelhantes a comandos de operações sobre ficheiros com algumas possíveis extensões para que seja possível assim escrever, ler e remover, entre outras. Sendo que as operações disponíveis serão:

Operação	Dados de Entrada	Resultado
PUT	String e Byte[]	Sucesso ou insucesso
GET	String	Byte[]
REMOVE	String	Sucesso ou insucesso
LIST	String	String[]
MKDIR	String	Sucesso ou insucesso
SEARCH	String	String[]

Tabela 3.1: Operações disponíveis pela API

Assim deste modo uma escrita de um ficheiro (PUT) ira necessitar do conteúdo do mesmo como também do nome pelo qual será identificado no sistema de armazenamento remoto. A operação de leitura (GET) ira devolver uma sequência de bytes sendo que lhe é fornecido o identificador de um ficheiro existente. A remoção ira funcionar de modo semelhante para um dado identificador. Por fim a operação de listagem permitira que para um dado identificador se permita listar informação relevante, nomeadamente como se trata de um sistema de ficheiros remotos, a criação de diretorias consiste numa vertente mui-

to apeladora, permitindo assim que o utilizador usando o comando de criação (MKDIR) crie as diretorias desejáveis.

IndexManager: Este módulo, ira armazenar e gerir informação relativa aos ficheiros guardados pelo middleware. Assim quando se deseja efetuar uma operação, seja ela de leitura ou de escrita neste modulo encontrar-se-á informação sobre o ficheiro em questão caso ele já existe e em que repositórios se encontra alojados, visto que para uma operação de leitura, a informação de repositórios usados servira para depois obter recorrendo ao **profiler** quais a melhor *cloud* para a operação. Servira para guardar também, para além de informação de replicação, informação sobre o número de fragmentos de um ficheiro como também o tamanho do mesmo. O índice devera ser remoto sendo que deste modo permitira com que o sistema possa ser usado em múltiplos dispositivos ou para casos de tolerância a falhas, deste modo, como se pode observar na figura o sistema permitira duas maneiras de operação. Operação de índice interno, sendo que nesta situação os dados e informação do índice encontram-se alojados nos próprios provedores recorrendo aos mecanismos oferecidos pelo *middleware.* Ou poderá utilizar repositórios externos, sendo esses existentes na cloud como serviço, nomeadamente o sistema OpenReplica. Sendo que colocando em qualquer destes modos o índice remotamente, permite-se a outras instâncias de *middleware* coexistirem e operarem sobre os mesmos dados. Sendo que este componente não poderá criar um único ponto de falta para o sistema. Para efeitos de otimização de acessos, usa-se um componente local de *caching,* componente este que ira permitir guardar informação recente sobre elementos acedidos, permitindo que a informação seja obtida em menor tempo e sem obrigar a potenciais custos acréscimos de aceder ao índice remoto.

Profiler: Este componente tem como objetivo endereçar as questões de redução de custos e redução de latências. Para permitir que este módulo funcione concretamente, o mesmo devera recolher métricas importantes e efetuar processamento relacionado com os ficheiros. Nomeadamente devera conseguir com uma dada operação manter informação sobre quais ficheiros são acedidos com mais frequência e quais não, sendo que com base nessa informação devera a certa altura transferir os que não são acedidos frequentemente para provedores que sejam mais rentáveis para armazenar ficheiros com poucos acessos, mantendo assim os ficheiros de uso frequente em provedores com tarifários mais baratos para acessos constantes. Por outro lado devera recolher informação sobre o desempenho das *clouds,* desempenho esse que caso o modo esteja em redução de tempos de resposta, per-

mitira com que a escolha de *clouds* seja a mais rápida para as operações em questão. A semelhança do **IndexManager**, toda a informação recolhida também devera ser armazenada remotamente para evitar pontos únicos de falha.

SecurityManager: Este componente é composto por subcomponentes com cada um deles com a sua função bem definida e importante. Para endereçar as questões de confidencialidade, temos o componente de encriptação que recorrendo a técnicas bem conhecidas e divulgadas de criptografia permite que um dado conteúdo seja ele ficheiro ou sequência de bytes seja encriptado face a um algoritmo com a sua respetiva chave ou então desencriptar o mesmo, criando assim ficheiros prontos para serem lidos pelo cliente no caso de leitura ou ficheiros prontos para serem submetidos. Para endereçar questões de integridade do conteúdo armazenado, existe neste componente o modulo Integrity. Este módulo recebe um ficheiro a ser submetido, calcula o código de integridade com base no seu conteúdo e devolve-o ao invocador. Esse código é importante para uso futuro, nomeadamente quando se deseja obter um ficheiro, permitindo verificar se o conteúdo do mesmo não foi alterado quanto residente na *cloud*. Por fim este componente tem adicionalmente um componente de autenticação, que seu objetivo é criar um mecanismo de verificar que um certo ficheiro foi mesmo criado pelo utilizador mencionado.

DataManager: Este módulo poderá ser visualizado como o componente central do *middleware*. Componente esse que ira receber pedidos efetuados a API anteriormente descrita e que consoante o pedido ira reunir os recursos necessários do sistema e contactar os restantes módulos, sendo este uma espécie de coordenador e controlador de operações. Mais concretamente após a receção de um pedido devera consultar o índice para obter informação existente do ficheiro em questão, em seguida devera consultar o **profiler** para que o mesmo lhe indique quais os provedores indicados para a operação e em seguida numa operação de escrita coordenar o sistema de segurança para produzir o resultado final a entregar as clouds, ou caso de leitura contactar as clouds e recuperar o conteúdo desencriptando em seguida e entregando ao utilizador. Consoante as informações indicadas pelo módulo de segurança devera também prosseguir com operações de recuperação de conteúdo sendo que poderão haver casos de falha de integridade. Adicionalmente este módulo devera conter operações que permitam fragmentar os dados a serem armazenados, permitindo distribuição mais refinada por múltiplas clouds. Sendo assim importante em aspetos de recuperação de dados que foram sujeitos a ataques, reduzindo possível necessidade de transferências adicional de dados.

3.3.4. Fluxos de processamento suportados na arquitetura de *software*

Tendo como referencia a visão introdutória dos principais componentes da arquitetura de middleware, apresenta-se de seguida como se suportam os fluxos de execução suportados pelos serviços disponibilizados pelos referidos componentes. Para este efeito, estes fluxos são ilustrados a partir das operações disponibilizadas pela API externa para suporte das operações dessa API. Os diagramas seguintes seguem a disposição de cores semelhante as cores usadas no diagrama de arquitetura (Figura 3.1) para uma fácil perceção de componentes envolvidos no respetivo processo.

3.3.4.1 Escrita de um objeto (PUT)

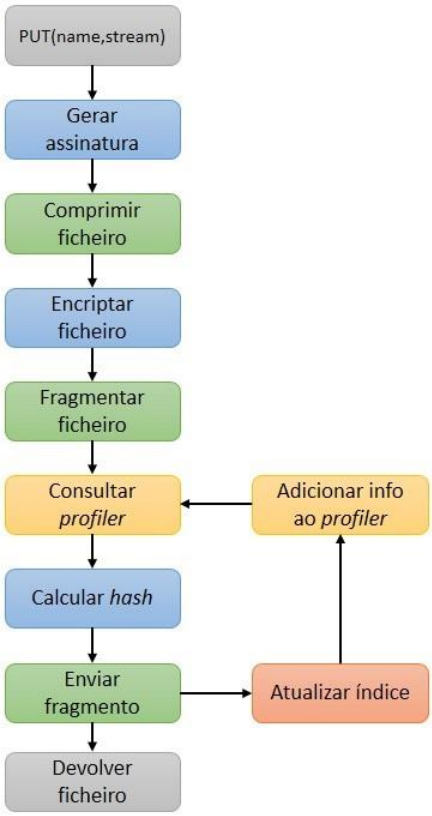

Figura 3.5: Diagrama da operação de PUT

Operação 1 Operação de PUT

```
1: procedure PUT(filename, file)
2:     aquire(lock)
3:     indexItem ← getIndexItem(filename)
4:     size ← length(file)
5:     signature ← getSignature(file,user_private_key)
6:     compressedData ← compress(file)
7:     encryptData ← encrypt(compressedData)
8:     numberOfBlocks ← smallestIntegerValueGreaterThan(size/blockSize)
9:     bytesRead ← 0
10:    for i ← 0 to numberOfBlocks do
11:        if bytesRead + blockSize > length(value) then
12:            bytes ← value[bytesRead..(length(value) - bytesRead)]
13:        else
14:            bytes ← value[bytesRead..(length(value) - blockSize)]
15:        end if
16:        readSize ← length(bytes)
17:        container ← createContainer(bytes)
18:        hash ← hash(bytes)
19:        addDataToContainer(container, signature, encryptionSettings, hash)
20:        providerList ← getBestProviders(PUT, blockSize)
21:        for c in providerList do
22:            uploadData(container,c)
23:            updateProfiler(blockSize,c)
24:            indexItem ← updateItem(indexItem,c)
25:        end for
26:    end for
27:    updateIndex(filename,indexItem)
28:    release(lock)
29: end procedure
```

Figura 3.6: Pseudocódigo da operação PUT

Como podemos observar no pseudocódigo anterior, a operação de escrita de um ficheiro nas diversas clouds usadas. Inicialmente trancamos as clouds seguindo assim o modelo de escritas de "*many readers – one writer*" garantindo que não há colisões de escritas nas clouds. Calcula-se a assinatura do conteúdo com base na chave privada do utilizador, comprime-se o conteúdo e cifra-se com cifras simétricas. Em seguida criam-se os diversos fragmentos dos dados com base no tamanho especificado. Para cada um deles, cria-se um fragmento onde colocam-se os dados, como também meta-informação relevante com os dados, sendo ela as cifras usadas e os códigos de integridade do fragmento em si e também a assinatura do ficheiro global. Consulta-se o Profiler para obter uma lista de fornecedores. Tendo essa lista, efetua-se um envio a cada um deles o contudo do fragmento sendo também atualizada a informação do Profiler para uso futuro da mesma.

3.3.4.2 Leitura de um objeto (GET)

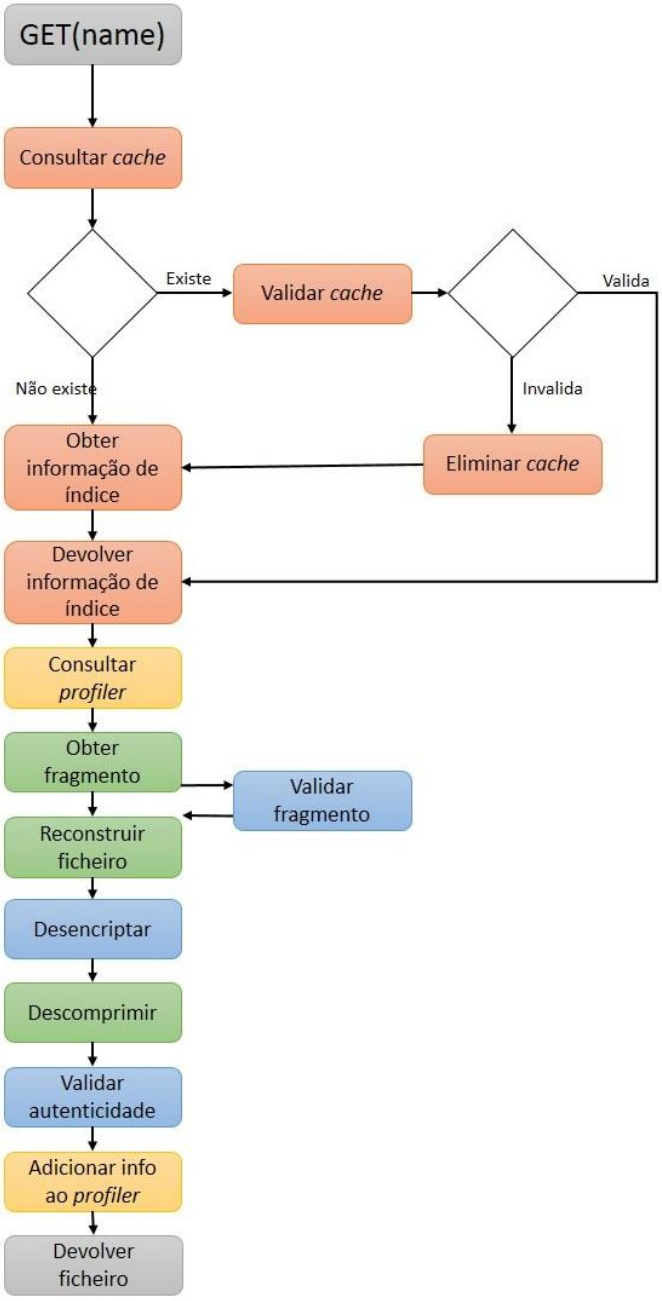

Figura 3.7: Diagrama da operação de GET

Operação 2 Operação de GET

```
1: procedure GET(filename)
2:     indexItem ← getIndexItem(filename)
3:     numberOfBlocks ← blocks(indexItem)
4:     availableProviders ← getProviders(indexItem)
5:     for i ← 0 to numberOfBlocks do
6:         providerList ← getBestProvidersFrom,(GET, blockSize, availableProviders)
7:         for connector in providerList do
8:             fragment[i] ← download(fragment_i,connector)
9:             updateProfiler(blockSize,c)
10:            isNotCorrupted ← verifyHash(data)
11:            if isNotCorrupted do
12:                break
13:            end if
14:        end for
15:    end for
16:    file ← reconstructFragments(numberOfBlocks,fragment[])
17:    decrypted ← decrypt(file)
18:    file ← decompress(decrypted)
19:    isAuthentic ← vefirySignature(file)
20:    if isAuthentic do
21:        return file
22:    else do
23:        return ERROR
24:    end if
25: end procedure
```

Figura 3.8: Pseudocódigo da operação GET

A semelhança da operação de escrita (PUT) a operação de leitura (GET) prossegue em linhas gerais do modo inverso, mas limita-se a uma cloud com sucesso, mais concretamente, contacta-se o índice para obter a informação relevante a localização dos fragmentos, contacta-se o Profiler para escolher um conjunto de melhores conectores nos quais se encontram as replicas. Em seguida contacta-se o melhor e obtém-se o fragmento, fragmento esse que é verificado a sua integridade e pronto para que após a obtenção de todos os fragmentos restantes seja reconstruído o ficheiro final. Após a reconstrução do ficheiro, descomprime-se e decifra-se sendo que depois é feita a última verificação de autenticidade antes de entregar o ficheiro.

3.3.4.3 Remoção de um objeto (DELETE)

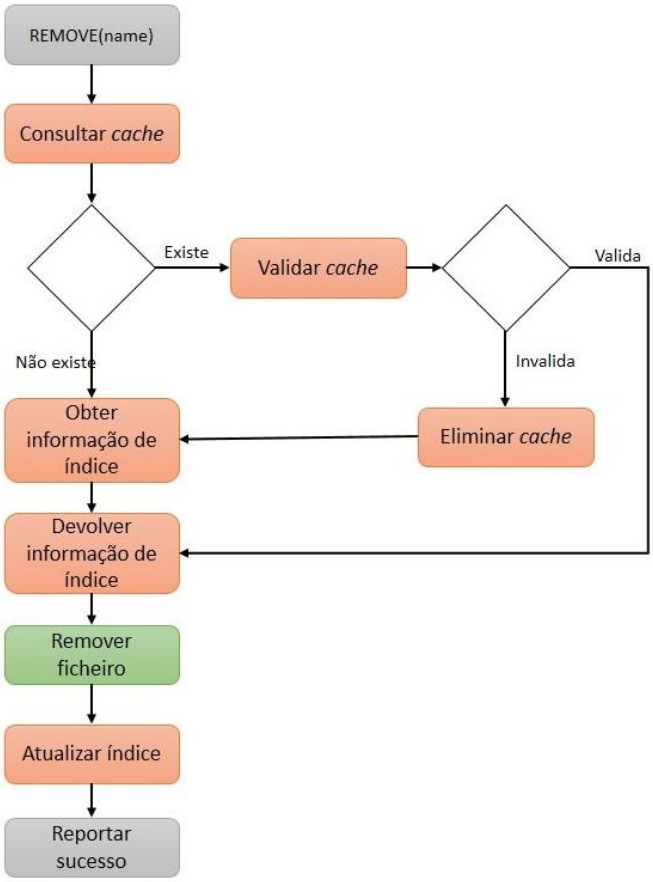

Figura 3.9: Diagrama da operação DELETE

Operação 3 Operação de DELETE

```
1: procedure DELETE(filename)
2:     indexItem ← getIndexItem(filename)
3:     numberOfBlocks ← blocks(indexItem)
4:     availableProviders ← getProviders(indexItem)
5:     for i ← 0 to numberOfBlocks do
6:         for connector in providerList do
7:             delete(fragment_i)
8:             updateProfiler(DELETE, connector)
9:         end for
10:    end for
11:    removeFromIndex(filename)
12: end procedure
```

Figura 3.10: Pseudocódigo da operação DELETE

A operação de remoção de conteúdo inicia-se obtendo toda a informação respetiva ao objeto que pretendemos eliminar. Por cada fragmento contactam-se os conectores respetivos e elimina-

se o conteúdo nesses armazenados. Por fim atualiza-se o índice eliminando as entradas dos ficheiros eliminados

3.3.4.4 Listagem de objetos numa diretoria (LIST)

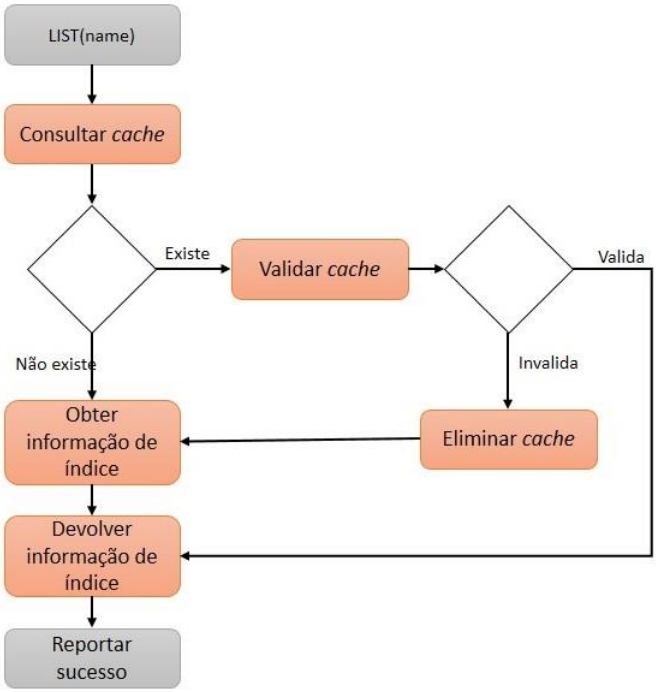

Figura 3.11: Diagrama da operação LIST

Operação 4 Operação de LIST

```
1: procedure LIST(directory)
2:     indexItem ← getIndexItem(directory)
3:     childElements ← getChildsOf(indexItem)
4:     return childElements
5: end procedure
```

Figura 3.12: Pseudocódigo da operação LIST

A operação de listagem de uma diretoria limita-se a consulta do índice não interagindo diretamente com os módulos do middleware. Sendo assim obtém-se a informação de lista em relação ao ficheiro em questão que depois é devolvida ao utilizador.

3.3.4.5 Criação de diretorias (MKDIR)

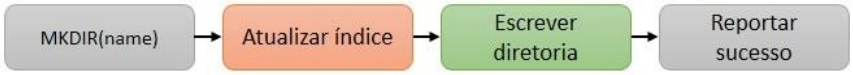

Figura 3.13: Diagrama da operação MKDIR

Operação 5 Operação de MKDIR

```
1: procedure MKDIR(directory)
2:     indexItem ← getIndexItem(parentDirectory)
3:     createDirectory(directory)
4: end procedure
```

Figura 3.14: Pseudocódigo da operação MKDIR

A semelhança da listagem de uma diretoria a criação também limita-se a criar uma entrada no módulo de indexação sendo que unicamente procura-se pela diretoria parente na qual adiciona-se a entrada da nova diretoria.

3.3.4.6 Pesquisa de objetos com base em meta-dados (SEARCH)

Figura 3.15: Diagrama da operação SEARCH

Operação 6 Operação de SEARCH

```
1: procedure SEARCH(filename, directory)
2:     itemList ← LIST(directory)
3:     result ←[]
4:     for item in itemList do
5:         if item contains filename do
6:             addtoList(result, item)
7:         end if
8:         if isDirectory(item) do
9:             list ← SEARCH(filename, item)
10:            addtoList(result, list)
11:        end if
12:    return result
13: end procedure
```

Figura 3.16: Pseudocódigo da operação SEARCH

A operação de pesquisa acima apresentada efetua-se de modo recursivo. Sendo que por cada invocação obtém-se recorrendo ao método de listagem todos os elementos da diretoria passada em argumento. Caso exista elemento que verifique o critério de pesquisa adiciona-se a lista de resultados, em seguida caso seja uma diretoria invoca-se de novo o algoritmo de pesquisa para a nova diretoria com o mesmo critério, ate por fim ter a lista de todas os valores onde o critério se verifique.

Passaremos a apresentar, de forma mais detalhada, os aspetos de modelação e suporte de execução de cada um dos módulos anteriormente apresentados, tendo em conta a sua estrutura interna

3.4. Modelação e suporte dos componentes da arquitetura de software

3.4.1. Módulo de Suporte da API externa

Este modulo tem como objetivo fornecer uma maneira simples e uniforme para operar o middleware. O seu objetivo é recorrendo a um conjunto de comandos definidos e disponíveis publicamente permitir a outras aplicações sejam elas locais ou remotas acederem ao middleware e invocarem as operações pretendidas. Operações essas previamente apresentadas anteriormente, as quais são, escrita (PUT), leitura (GET), remoção (DELETE), criação de diretorias (MKDIR) e pesquisas por palavras-chave (CHDIR) como também é importante oferecer acessos a outro tipo de operações nomeadamente acessos a recolha de dados estatísticos ou recarregamento dinâmico de configurações do sistema. Importante neste módulo é permitir que haja uma resolução de questões de heterogeneidade de implementações permitindo abstrair as restrições impostas pela linguagem desenvolvida, logo o sistema de gestão de API, sistema esse que devera traduzir as invocações externas para invocações internas ao API e devolver o conteúdo, será encar-

regado de fornecer esse conjunto de funcionalidades e propriedades, atender aos pedidos externos, traduzi-los para pedidos internos e devolver finalmente o resultado ao invocador externo.

3.4.2. Módulo de indexação

Todos os dados armazenados nesta arquitetura devem ser guardados numa estrutura fácil para o *middleware* numa outra dada altura poder facilmente aceder aos conteúdos como também rápida. Assim é necessário utilizar um módulo de indexação. Assim com objetivo a manter os acessos aos dados no mínimo possível o índice deve ser implementado para que seja possível obter a informação estritamente necessária. Logo quando se pretende aceder a raiz e listar os seus conteúdos à priori não necessitamos de aceder a informação completa do sistema, sendo essa informação detalhada de eventuais diretorias e informação de ficheiros. Assim este mecanismo ao listar uma diretoria em particular devera apresentar nomes de conteúdos sendo que se for necessário obter mais informação referente a um ficheiro então nesse caso devera aceder-se ao ficheiro de índice referente a esse ficheiro. Adicionalmente como o sistema permite replicação por diversos provedores como também possível fragmentação de dados, devera haver informação sobre esses conteúdos, nomeadamente o ficheiro referente a um objeto armazenado remotamente, devera conter o número e identificadores das réplicas e por cada réplica terá que ter associada uma lista de provedores onde ela se encontra alojada.

Importante mecanismo deste índice, como mencionado anterior, é um subcomponente auxiliar, que seu objetivo é reduzir os acessos remotos sendo que recorre a informação colecionada durante a execução do mesmo, nomeadamente uma *cache*. Assim quando se pretende efetuar uma leitura de um ficheiro, primeiro consulta-se a *cache* local, e caso exista evita-se de ter que ir ao índice remoto. Caso contrário seremos obrigados a consultar as *clouds*.

3.4.2.1 Modelação do componente

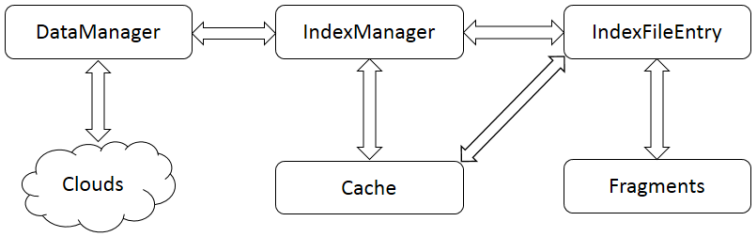

Figura 3.17: Estrutura do componente

3.4.3. Módulo de Profiling (Profiler)

Importante componente da arquitetura apresentada sendo ele uma vertente pouco explorada, é o sistema de criação e gestão de perfis de conteúdos e de provedores. Nesta secção serão abordadas em detalhe as métricas que serão utilizadas para o cálculo e decisão do melhor provedor de para uma dada operação num dado momento.

3.4.3.1 Métricas importantes a endereçar

Existem diversas métricas que podem ser endereçadas, entanto consideramos que para efeitos desta tese um pequeno conjunto será considerado importante. Mais concretamente existem duas grandes categorias: Custo financeiro e latências. No que toca a custos financeiros devido a um grande leque de modelos de aplicação de tarifários por parte das diversas operadoras consideram-se importantes os custos associados com a quantidade de dados armazenados, o custo de operações, sendo elas escritas, leituras ou remoção de conteúdo, como também o tráfego efetuado por cada operação distinta. Custos esses que em combinação entre eles, permite ao sistema oferecer informação sobre qual a mais barata para a operação pretendida.

A outra categoria de métricas importantes é em relação as métricas de latência, métricas essas associadas a escrita ou leitura de um conteúdo sendo diferenciadas, como também métricas das restantes operações. Métricas essas recolhidas durante as operações do middleware nomeadamente recolha de tempos de execução. Sendo que a partir dessas métricas seja possível escolher o conjunto de clouds mais indicado para a operação em questão.

3.4.3.2 Mecanismos de *profiling*

Com visão as métricas importantes anteriormente apresentadas deveram existir dois mecanismos distintos de cálculo e decisão de qual a melhor cloud a usar. Um mecanismo que devera calcular as métricas relevantes aos desempenhos de cada cloud e um mecanismo que deve calcular a frequência de uso e acesso.

O mecanismo de desempenho funciona em paralelo com o *middleware*, especificamente, quando um utilizador efetua uma operação de escrita são efetuadas escritas num subconjunto de clouds, assim calcula-se com face ao tempo de execução da escrita em si quantos bytes foram escritos, permitindo assim ter noção do desempenho das clouds. Por cada *cloud* acedida associa-se assim essa métrica que base na qual devera ser feita a escolha futura.

O mecanismo de custos funciona em paralelo com o middleware, sendo que para além de cálculo de métricas tem uma função de migração adicional, que será abordada mais a frente. Este mecanismo calcula, recorrendo ao mecanismo apresentado em [25], um valor associado a

cada ficheiro, valor esse como pode ser visto na Figura 3.18 obtido a partir do tamanho médio do sistema como também da frequência de acessos. Nesta questão entra uma tarefa periódica que com o passar do tempo calcula base nesse algoritmo a frequência de acessos.

Operação 6 Operação de getAge(filename)

```
1: procedure GETAGE(filename)
2:     indexItem ← getIndexItem(filename)
3:     size ← getSize(indexItem)
4:     averageSize ← getAverageSize()
5:     oldAge ← getPreviousAge(indexItem)
6:     lastAccess ← getLastAccessTime(indexItem)
7:     if lastAccess < now + interval do
8:             return oldAge + (average / size ) * 0.9
9:     else do
10:            return oldAge * 0.9
11:    end if
12: end procedure
```

Figura 3.18: Pseudocódigo de cálculo de métrica

3.4.3.3 Reposicionamento de Dados

Adicionalmente o componente de Profiler, a modos de reduzir custos de armazenamento de dados, pode com base no cálculo do valor anterior, reorganizar os dados de forma a reduzir ainda mais os custos ao utilizador. A reorganização contabiliza os custos adicionais da transferência, onde eles se aplicam, e caso os custos justifiquem a transferência seguindo os novos critérios, então é recalculado o novo conjunto de provedores nos quais os dados serão colocados. O critério de escolha se um dado ficheiro devera ser analisado para reposicionamento é feito com base no valor calculado sobre a frequência de acessos e com o valor limite estipulado pelo utilizador. Em seguida apresentamos o algoritmo que trata de reposicionar os dados.

Operação 8 Operação de migrate(filename)

```
1: procedure MIGRATE(filename)
2:     age ← getAge(filename)
3:     isArchived ← getArchiveStatus(filename)
4:     trigger ← getTrigger(configuration)
5:     migrate ← false
6:     if age > trigger and not isArchived do
7:         bestRepositories ← getCheapArchive(filename, providers)
8:         migrate ← true
9:     else if age < trigger and isArchived do
10:        bestRepositories ← getCheapAccess(filename, providers)
11:        migrate ← true
12:    end if
13:    if migrate do
14:        oldRepositories ← getRepositories(filename)
15:        newRepositories ← bestRepositories - oldRepositories
16:        unusedRepositories ← oldRepositories - bestRepositories
17:        success ← migrate(filename, oldRepositories, newRepositories)
18:        delete(unusedRepositories)
19:        if success do
20:            updateIndex(filename, bestRepositories)
21:            setArchiveStatus(filename, true)
22:        end if
23:    end if
24: end procedure
```

Figura 3.19: Pseudocódigo de reposicionamento

Calcula-se o conjunto de melhores provedores recorrendo a uma outros critérios de arquivamento, critérios esses declarados pelo utilizador. Após ter melhores provedores, efetua-se uma escrita nos quais não existem previamente os dados, com base na intercessão de listas de repositórios. Em seguida remove-se os dados dos provedores que já não fazem parte da lista de provedores para arquivar os dados. Após a operação ter sido concluída com êxito atualiza-se o índice. O contrário verifica-se caso a frequência de acesso justifique que um ficheiro seja removido de provedores mais apropriados para arquivamento para provedores mais indicados para acessos frequentes.

No final desta operação então os dados com pouco acesso estarão alojados em provedores com custos inferiores de armazenamento e os de dados de acesso frequente em provedores que tem custos de acessos reduzidos.

3.4.3.4 Modelação do componente

Na figura seguinte ilustramos as relações arquiteturais com foco central o modulo de Profiler e quais as subcomponentes do mesmo.

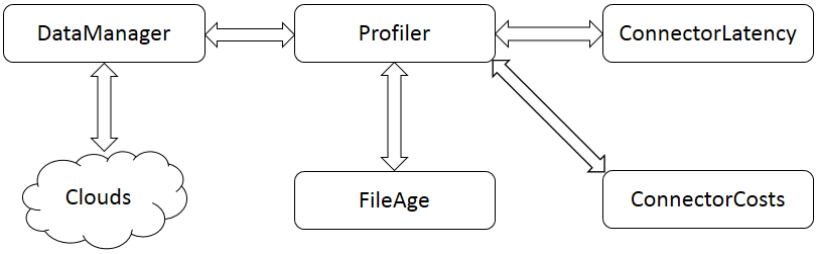

Figura 3.20: Modelação do Profiler

3.4.4. Aspetos de flexibilidade e sua configuração

Todos os componentes mas principalmente o de *profiling*, deverão ser concretizados de forma a serem modulares e flexíveis a adaptação de novas configurações, visto que existe, como podemos observar, com certa frequência atualizações de tarifários como também novos provedores a emergirem no mercado. Assim o sistema devera ser capaz de adaptar-se facilmente a essas novas alterações, mais concretamente será importante para a arquitetura alterar o comportamento do *profiler* facilmente. Desta forma a configuração tem que ser o mais possível modular e extensível pelos diversos provedores. Este mecanismo é importante para o sistema sendo que poderá ser alojado num serviço remoto (rede ou *cloud* computacional) a modos de alterar mesmo em tempos de execução as diversas configurações sem que haja impacto na disponibilidade do mesmo. Em suma os aspetos de flexibilidade cuja importância já foi anteriormente referida, são os componentes independentes entre si e modelares a modos, respeitando um conjunto de regras e gramáticas bem definidas permitir com que se associe facilmente a cada provedor a sua respetiva informação relevante para o *profiler*.

3.4.5. Processamento com informação de profiling

Periodicamente, para redução de custos, sendo o profiler configurado para esse modo, calcular-se-á uma atualização métrica apresentada anteriormente associada a cada ficheiro. Assim essa tarefa auxiliar, ao calcular o novo valor ira verificar se face a esse valor e com a configuração declarada anteriormente o ficheiro devera ser considerado para arquivar. Para arquivar consideramos os ficheiros que não são acedidos com muita frequência facto que torna-os candidatos a serem armazenados em provedores com custo inferior de armazenamento mas custo indiferenciado de escrita, tendo em conta que esse custo não seja proibitivo para armazenar. Assim neste modo quando se efetua uma operação de migração de ficheiros, consultam-se o *profiler* que por sua vez face ao tamanho de dados que se pretende escrever e transferir, calcula com base nos diversos tarifários um quórum de *clouds* cujo custo de armazenamento seja adequado, a semelhança de uma escrita de ficheiros que por defeito considera-se um ficheiro que poderá ter acessos logo, o *profiler* calcula as *clouds* com fator principal o custo de transferências.

Por outro lado, o *profiler* poderá funcionar em modo de latências, como anteriormente elaborado. Mais concretamente ao recolher as métricas e associa-las a cada uma *cloud,* realizara uma espécie de histórico de medidas. Medidas essas que servirão de critério para a escolha. Assim quando se efetua uma operação de escrita e deseja-se guardar os conteúdos nas *clouds* mais rápidas, o *profiler* em vez de criar uma lista com base nos custos, elabora essa lista com base nas latências e ritmos de processamento, sendo que é um histórico poderá fornecer informação mais precisa.

3.4.6. Módulo de Segurança (Security Manager)

Uns dos vários objetivos desta tese tem a ver com a segurança dos conteúdos que o utilizador opta por guardar e gerir recorrendo a esta middleware.

3.4.6.1 Modelo de adversário e serviços de segurança

Para efeitos desta tese consideramos que o modelo de adversário perante este middleware é composto pelos seguintes casos onde um atacante poderá:

- Alterar os conteúdos de um subconjunto dos fornecedores corrompendo assim os dados existentes, mas não poderá corromper todos ao mesmo tempo.
- Ter acesso ao sistema de ficheiros remoto a modos de poder aceder aos conteúdos armazenados no mesmo.
- Negar comunicação a um conjunto de clouds sendo que deste modo a dada cloud estará incontactável.
- Ter acesso aos conteúdos de uma cloud a fins de alterar o autor fazendo-se passar por um autor legítimo, sendo que poderá neste caso criar ficheiros falsos.

Nestes cenários acima mencionados o atacante poderá ser um terceiro estranho ao sistema que possa eventualmente ter acesso aos provedores em questão, ou lançar ataques de negação de serviço negando assim temporariamente acesso por parte do cliente aos conteúdos dele armazenados. Por outro lado o atacante poderá ser alguém da própria cloud, sendo questões de contracto, nomeadamente cenários onde por norma praticas empresariais ou de termos de contracto neguem efetivamente o acesso aos dados (*vendor-lock-in*) ou por parte de um próprio funcionário da empresa que efetue acessos ilícitos aos ficheiros guardados la.

3.4.6.2 Mecanismos de Segurança e Confiabilidade

Para endereçar o modelo de atacante apresentado anteriormente serão usados mecanismos focados a resolver essas problemáticas. Mais concretamente para evitar que um atacante tenha acesso aos conteúdos o módulo de segurança ira encriptar todos os conteúdos la colocados. Sendo que deste modo se um atacante não tiver a chave de encriptação não poderá obter acesso aos conteúdos.

3.4.6.3 Mecanismos de integridade e autenticidade

Para cumprir os objetivos de ataques de integridade sendo essas de alteração de conteúdo mesmo ele cifrado, ou para casos onde o atacante possa criar conteúdo malicioso sendo que esta deste modo a fazer-se passar por um utilizador legítimo, recorre-se a mecanismos bem definidos de integridade e autenticidade. Mais concretamente os dados que são armazenados após serem cifrados serão sujeitos a processos de cálculos de códigos de integridade, códigos esses que são anexados a cada um desses dados a modos que após a escrita de um utilizador, ao obter um dado conteúdo das diversas *clouds* será efetuada uma verificação no cliente com o conteúdo obtido face ao código de integridade armazenado remotamente, caso os códigos sejam idênticos assume-se que o conteúdo não foi sujeito a ataques de manipulação, podendo assim prosseguir com as restantes funcionalidades do *middleware* nomeadamente seguir com o processo de desencriptação do conteúdo.

Por outro lado para provar que o conteúdo final foi criado por um utilizador legítimo do sistema, sendo que poderá haver situações onde o sistema permitira múltiplos utilizadores, após a obtenção de um ficheiro, antes de se entregar ao cliente verifica-se recorrendo a mecanismos de autenticidade de conteúdo. Nomeadamente recorrendo a técnicas de criptografia assimétrica, onde cifrando com a chave privada de um utilizador, depois pode-se verificar a autenticidade recorrendo puramente a chave pública, sendo que deste modo o autor não expõe a sua chave privada. Ao não expor as chaves privadas fornece-se um nível de segurança adicional, sendo que pelas diversas instâncias do *middleware* poderá só haver a chave pública distribuída.

3.4.6.4 Mecanismos de Controlo de Acesso

Ao utilizar chaves públicas podemos assegurar que o autor de um ficheiro prova ser quem diz. Deste modo para garantir controlo de acesso, referente a autores de ficheiros, a utilização de assinaturas com base em chave privada são suficientes. Para garantir controlo de acessos é possível usar um esquema onde a chave simétrica que foi usada para cifrar o contentor pode ser incluída no cabeçalho sendo previamente cifrada com uma chave pública de um utilizador ao

qual se pretende dar acesso ao conteúdo. Desta forma ao utilizar a versatilidade do cabeçalho podemos dar acesso ao ficheiro a um conjunto de utilizadores incluindo assim previamente os dados necessários no cabeçalho.

3.4.6.5 Modelação do componente

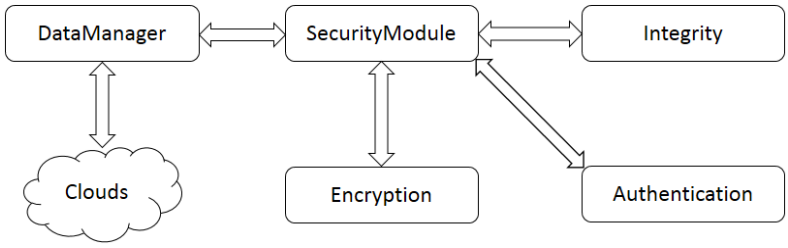

Figura 3.21: Modelação do componente de Segurança

Assim como podemos observar, na figura anterior o componente de segurança coordena as operações de encriptação, autenticação e integridade como também consulta os respetivos subcomponentes para validar operações anteriores, nomeadamente verificar a integridade de um objeto face a um código ou verificar a autenticidade como também desencriptar um dado conjunto de dados.

3.4.7. Modelo de dados e conceção do módulo de gestão de dados (DataManager)

Como anteriormente abordado o modelo de dados do sistema é composto por uma estrutura específica. Estrutura essa composta por um conjunto de dados que podem representar uma parte de um ficheiro sendo este um fragmento ou um ficheiro completo, e um cabeçalho, cabeçalho esse que devera abordar a heterogeneidade e diferentes módulos do próprio middleware. Assim para esse efeito o cabeçalho será um mapa o qual estará indexado por uma chave para cada valor armazenado nele. Deste modo permite-se que seja simples e abstrata a operação sobre os cabeçalhos, mais a frente será abordado em maior profundidade este tipo de contentor e o seu cabeçalho em particularidade.

Estes contentores são criados e geridos pelo módulo de dados (DataManager), módulo este que ira coordenar os acessos por parte dos módulos sendo que cada módulo poderá colocar informação no seu cabeçalho sobre algoritmos e aspetos de configuração importante para quando um ficheiro for lido ou recuperado no futuro.

3.4.8. Encapsulamento de dados e fragmentos para armazenamento em nuvem

Para combater a heterogeneidade dos provedores de armazenamento no que toca a questões de formatação e particularidade de gestão de dados por parte do middleware necessita-se de um componente que permita abstrair essas particularidades perante os restantes componentes do middleware. Deste modo necessita-se de uma estrutura específica para endereçar esta questão em particular. Adicionalmente importante fator de compatibilidade desta estrutura é permitir a todos os componentes do middleware acederem a ela e utilizarem-na sem que haja restrições de desenho impostas. Assim temos a noção de contentores, contentores esses que terão qualquer tipo de dados contido neles, seja um ficheiro inteiro ou parte do mesmo, nomeadamente um fragmento. Em seguida abordamos as particularidades e modelação destes contentores e como são geridos e mantidos pela arquitetura.

3.4.8.1 Modelação de DATA-CONTAINERS

Como abordado anteriormente este contentor devera ser genérico e simples de ligar sendo que terá que garantir abstração da heterogeneidade dos provedores permitindo aos componentes do middleware acederem a este. Assim este componente será composto por 2 partes como pode ser visto na figura seguinte, pelos dados em si, e pelo cabeçalho. O cabeçalho desta estrutura de dados devera ser feita a permitir que todos os componentes do sistema possam facilmente operar sobre ela permitindo uma abstração entre componentes. Logo para garantir este tipo de características o cabeçalho é para efeitos de customização uma estrutura par chave-valor, onde cada componente pode adicionar entradas para uma respetiva chave. Permite-se assim que cada consiga aceder aos dados de importância atribuídos a cada ficheiro ou fragmento armazenado na cloud.

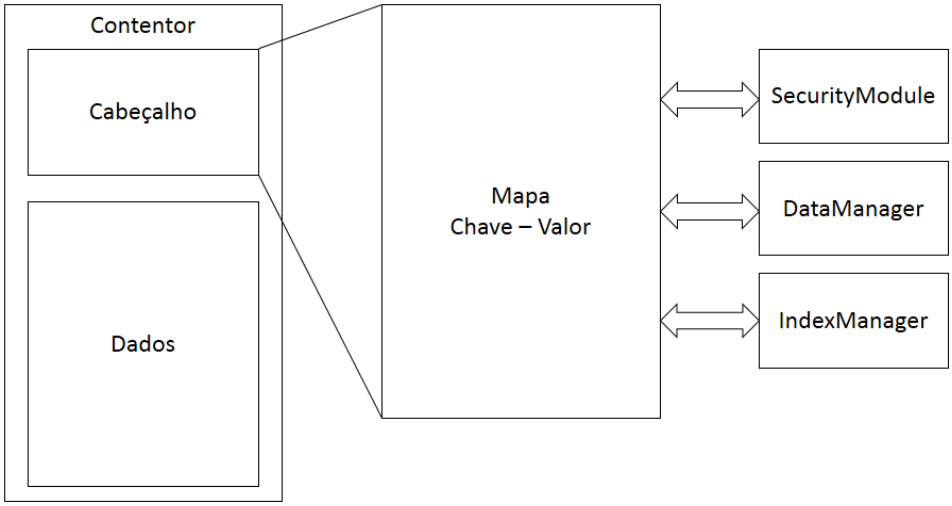

Figura 3.22: Cabeçalho do contentor

3.4.8.2 Indexação de DATA-CONTAINERS

Estes diversos fragmentos devem ser fáceis de identificar e ordenar entre si, nomeadamente em questões de reconstrução do ficheiro a apresentar pelo middleware. Assim como apresentado acima o gestor de índice para além de guardar internamente identificadores e valores respetivos a cada fragmento nomeadamente quais são os fragmentos que compõem um ficheiro e onde eles se encontram. Cada fragmento tem em si assim, um valor referente a que parte do ficheiro representa. Adicionalmente pode-se por parte deste módulo adicionar informação adicional em relação ao ficheiro que representa sendo possível assim a partir só do cabeçalho poder encontrar os restantes fragmentos e em que conectores se encontram alojados como também os nomes dos mesmos. Sendo que uma estrutura chave-valor permite tal informação a indexar.

3.4.8.3 Proteção de DATA-CONTAINERS

Os conteúdos do cabeçalho como qualquer outra parte do ficheiro podem ser sujeitos a ataques de manipulação de conteúdo. Assim durante as operações de verificação de integridade, para questões de maior segurança e integridade quando se calcula os códigos de integridade dos dados adiciona-se também os campos do cabeçalho, sendo que um possível ataque ao conteúdo não se limite só aos dados mas ao cabeçalho também. Os contentores de dados poderão estar cifrados com chaves diferentes entre si e com algoritmos diferentes, para este efeito o componente de encriptação devera usar os campos que forem mais adequados para ele no mapa anteriormente abordado para indicar informação sobre as técnicas de encriptação e chaves usadas permitindo que as mesmas não sejam expostas a terceiros ou mesmo aos provedores das clouds. Este fator permite que para além de cada fragmento ter a sua chave pode-se facilmente alterar as configurações por parte dos componentes atuais no middleware e como os fragmentos armaze-

nados anteriormente contem informação por parte dos algoritmos usados permite que o sistema possa usar a informação de configuração do cabeçalho, sendo assim possível recuperar ficheiros antigos de outras configurações sem que haja incompatibilidade por parte do middleware referente a algoritmos de encriptação, integridade e autenticidade.

3.4.8.4 Processamento de DATA-CONTAINERS

Como abordado anteriormente, os contentores são armazenados em clouds remotas e compostos por um cabeçalho no qual cada componente armazena os dados importantes relacionados com o ficheiro ou o fragmento nele armazenado. Deste modo os módulos atribuem dados durante a sua operação ao cabeçalho, nomeadamente os códigos de integridade, as configurações de encriptação usadas para depois permitir desencriptar, como também identificadores de criador e de sequência de fragmentos. Quando os dados colocados no cabeçalho estão prontos para ser escritos no ficheiro, gera-se uma sequência de bytes que represente o byte a modos de ser facilmente reconstruído quando se lê e coloca-se no início do ficheiro que se pretende escrever. Quando se lê esse contentor, extrai-se do início os dados que representam o mapa e reconstrói-se a mesmo a fins de permitir aos componentes poderem aceder aos valores previamente colocados la. O processamento é efetuado da mesma forma que um acesso a uma estrutura de um mapa, mapa o qual depois é sujeito a operações de transformação e reconstrução para um formato facilmente armazenável.

3.5. Generalizações do modelo e arquitetura de software

O sistema, tal como se encontra definido na secção 3.2 pressupõe uma instanciação do mesmo num servidor confiável. Todas as propriedades de segurança e confiabilidade da solução são garantidas e o acesso é intermediado por uma instância do middleware descrito.

3.5.1. Instanciação da Arquitetura do Sistema e Variantes de Concretização

A arquitetura de referência inicialmente apresentada poderá ser concretizada em diferentes variantes, tendo em vista serem suportadas em diferentes tipos de ambiente. Com efeito, a modularidade dos componentes da arquitetura base de referência permite que a sua concretização em diversas variantes permita endereçar de forma diferenciada as possíveis vantagens e desvantagens de diferentes instanciações do modelo de sistema em diversas implementações.

Nas secções seguintes serão apresentadas as diferentes variantes de instanciação do modelo e arquitetura do sistema proposto, tendo em vista diferentes cenários de concretização e suportes de execução. As variantes apresentadas são as seguintes:

- Variante *Middleware* Local
- Variante *Middleware* com Arquitetura em *Proxy*
- Variante *Middleware Proxy* com replicação
- Variante *Middleware* como serviço na nuvem ou *cloud-based middleware as a service* (*MaaS*)

Tendo por base a arquitetura de referência anteriormente apresentada, cada uma das variantes é caracterizada na sua arquitetura e aproximação à sua concretização, discutindo-se vantagens, dificuldades ou inconvenientes dessa concretização, tendo em conta as seguintes dimensões de análise:

- Endereçamento de propriedades de segurança
- Endereçamento de requisitos de confiabilidade
- Complexidade da implementação
- Componentes adicionais específicos na arquitetura de referência

3.5.2. Variante Middleware Local

3.5.2.1 Caracterização desta variante

Neste modelo o *middleware* encontra-se alojado na máquina do cliente, como pode ser observado na Figura 3.23. Assim o dispositivo do cliente liga-se diretamente as clouds, com o cliente a efetuar invocações locais ao *middleware* sendo via essas invocações efetuadas por aplicações recorrendo a API disponibilizada pelo respetivo modulo. Sendo que nesta situação o computador local do cliente deve ser confiável e seguro.

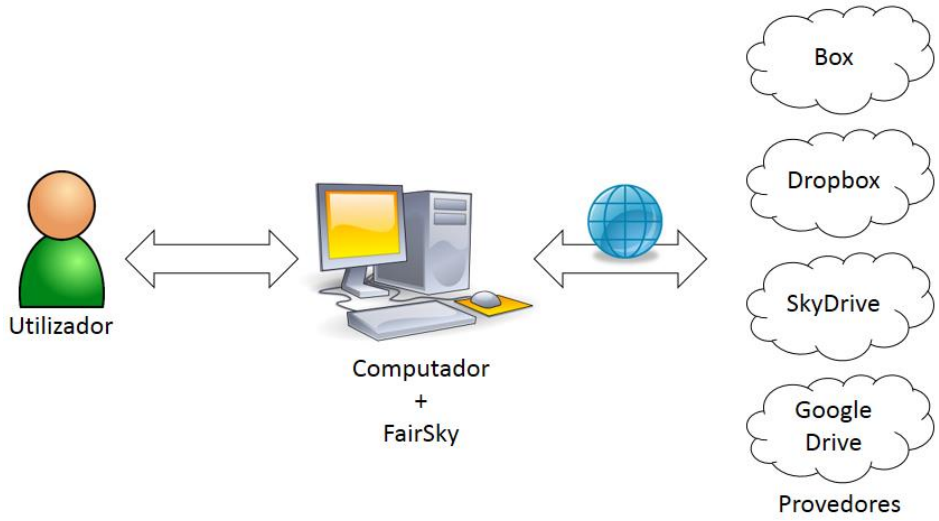

Figura 3.23: Cliente local

3.5.2.2 Concretização desta variante

Esta variante, encontra-se puramente alojada no dispositivo do cliente, como anteriormente abordado, sendo que todas as comunicações de e para os provedores são efetuadas desse mesmo dispositivo.

3.5.2.3 Disponibilidade e Recuperação do Estado de Execução

Esta variante esta limitada a disponibilidade da aplicação do utilizador, nomeadamente, o *middleware* devera ser invocado pelo utilizador ou instalado como serviço, e esta restrito a conectividade do dispositivo as diversas clouds. Em caso de falha o sistema é robusto e auto contido, sendo que qualquer falha ao nível da variante enquadra-se no modelo de falhas e atacante proposto anteriormente, sem incluir falhas que impossibilitam o funcionamento do sistema FairSky, por exemplo, situações onde não existe conectividade a Internet ou falha técnica por parte do dispositivo.

3.5.2.4 Vantagens e inconvenientes

A variante local tem um certo conjunto de vantagens e inconvenientes no seu uso face a outro tipo de soluções, mais concretamente as vantagens são: acesso por um único cliente, o módulo de *profiler* permite que a otimização de acessos em questões de tempo se adeque melhor para um utilizador móvel, nomeadamente situações onde o cliente pode aceder ao middleware a partir de diversos locais geográficos, mantendo assim um melhor desempenho de acessos, sendo também uma possível vantagem o facto de o sistema estar ativo somente quando o cliente dispõe do dispositivo. Por outro lado, durante as operações de escrita nas diversas clouds e com um número elevado de replicação, o sistema terá que comunicar com todas os provedores sendo que assim poderá utilizar largura de banda em demasia para um utilizador local.

3.5.3. Variante Middleware com Arquitetura em Proxy

3.5.3.1 Caracterização desta variante

Ao contrário do modelo anterior, este encontra-se alojado num computador numa rede local como na Figura 3.24. Deste modo o *middleware* age como um serviço de rede, sendo os clientes outros dispositivos existentes na rede local. Esses dispositivos podem ser de natureza variada, mais concretamente, repositórios de dados na rede, aplicações residentes nos clientes que invocam as operações recorrendo a API do *middleware*, ou mesmo um serviço de armazenamento fornecido pelo computador que aloja o *middleware*, como por exemplo um serviço de NAS cujo espaço real de armazenamento. Sendo que nesta situação, ao contrário da anterior, e consoante a

aplicação a usar o *middleware*, poderão existir necessidades adicionais de segurança por parte de todos os participantes da rede, visto todos terem acesso a esse serviço.

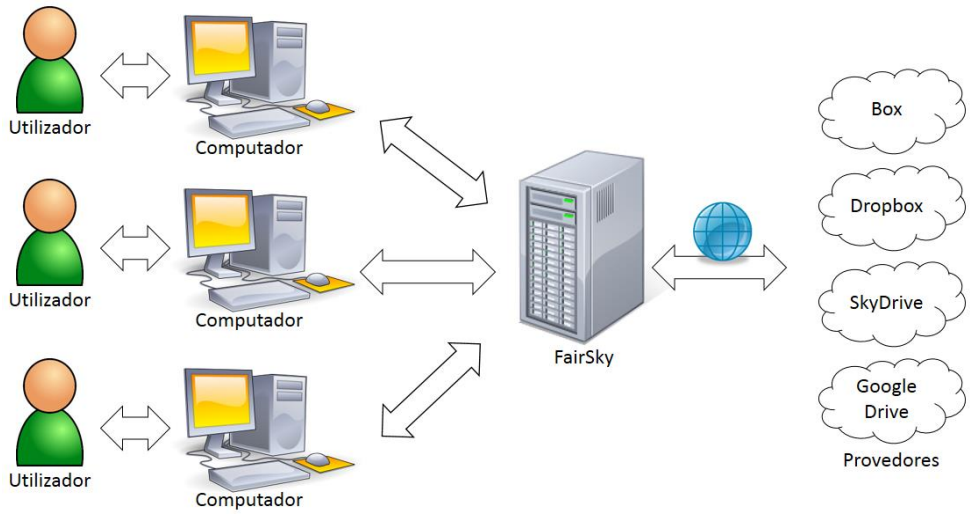

Figura 3.24: Variante Proxy

3.5.3.2 Concretização desta variante

Será necessário um servidor dedicado na rede local onde todos os clientes da mesma poderão aceder a uma API de rede para efetuarem as operações, sendo que neste caso os clientes enviam e recebem os ficheiros dentro da própria rede e o servidor *proxy* interage com as clouds em si fornecendo assim uma noção de repositório local. Assim é necessário uma infraestrutura adicional em comparação com a variante anterior.

3.5.3.3 Disponibilidade e Recuperação do Estado de Execução

Esta variante encontra-se alojada num dispositivo dedicado que atende pedidos efetuados de muitos computadores numa dada rede local ao qual se encontra adjacente. Assim de modo semelhante a disponibilidade do sistema será idêntica a disponibilidade do servidor onde este se encontra alojado sendo também sujeito a disponibilidade e taxas de utilização da rede. A recuperação de dados segue de modo semelhante ao modelo de falhas e modelo de atacante proposto neste documento. Logo a aplicação com a arquitetura proposta esta unicamente dependente dos provedores.

3.5.3.4 Vantagens e inconvenientes

Esta variante tem um conjunto de vantagens e inconvenientes que se aplicam a questões associadas ao modelo de uso, nomeadamente as vantagens enquadram-se na disponibilidade do servidor que fornece acesso ao middleware, permitindo assim um serviço externo não dependendo do dispositivo do utilizador, também permite múltiplos utilizadores, isto é um repositório a ser

partilhado e usado por múltiplos utilizadores em simultâneo. Por outro lado existe uma necessidade de controlo de concorrência por parte das escritas dos diversos utilizadores, poderá ser inconveniente se muitos utilizadores estão a fazer escritas, sendo que os mesmos são colocados em espera ate o sistema de concorrência os permitir escrever, concretamente situações onde já encontra-se outro utilizador a escrever.

3.5.3.5 Generalização da arquitetura

Esta variante pode-se generalizar-se e abranger utilizadores que não se encontram obrigatoriamente na rede local do middleware sendo assim tornada num serviço de Internet com um único ponto de acesso. Deste modo permite-se que utilizadores possam aceder a partir de outros locais. Adicionalmente poderá ser executado como um middleware alojado num computador local a um utilizador, mas ao mesmo tempo fornecendo um ponto de acesso para os outros dispositivos da rede local, assim torna-se numa variante local e proxy em simultâneo.

3.5.4. Variante Middleware Proxy com replicação

3.5.4.1 Caracterização desta variante

Na seguinte figura, observamos uma variante adicional desta implementação, o uso de múltiplas instâncias do middleware em simultâneo sobre o mesmo conjunto de clouds, nomeadamente, em modo de replicação de serviços. As várias réplicas do middleware usam as clouds em si para coordenarem-se entre si, nomeadamente nesta versão o sistema comum de repositórios permite que o sistema seja coordenado com base nesses repositórios. Assim cada réplica coloca na cloud a informação de ela própria, permitindo as restantes durante operações de leitura e escrita consultarem essa informação e fornecer abstração ao utilizador do modo que os repositórios são geridos.

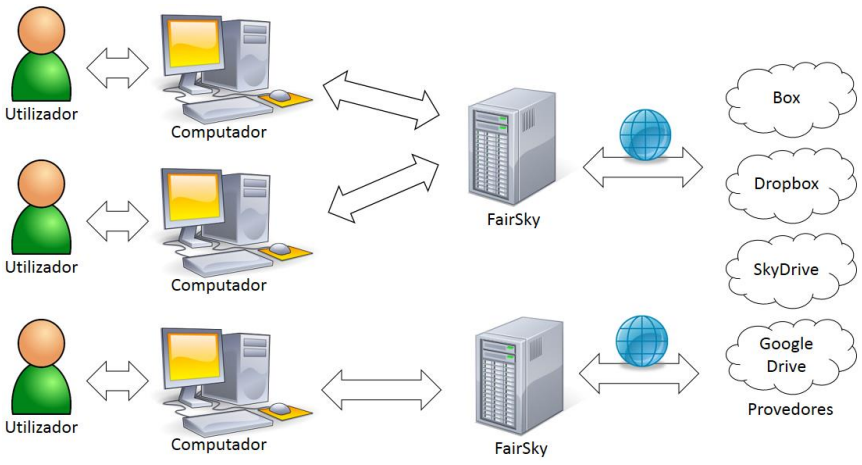

Figura 3.25: Variante Proxy com replicação

3.5.4.2 Concretização desta variante

Esta variante é muito semelhante a versão sem replicação só que existem múltiplos pontos de acesso ao repositório virtual. Serão necessários múltiplos dispositivos na rede local a alojarem o middleware, aos quais os diversos utilizadores conectam-se e efetuam os pedidos e operações, de modo semelhante a uma única instanciação do sistema FairSky.

3.5.4.3 Disponibilidade e Recuperação do Estado de Execução

Com o uso de réplicas do middleware permite evitar situações de único ponto de falhas. Assim com um dado número de réplicas do middleware permite-se que para além da tolerância a falhas de parte dos provedores, tolerância a falhas por parte do próprio middleware. Permitindo assim aos utilizadores acederem aos serviços com alto nível de disponibilidade. Para além da disponibilidade em questões de falhas, permite-se distribuição de carga pelas diversas réplicas proporcionando um maior número de clientes a serem servidos pelo sistema. Como todos os dados importantes para o funcionamento do sistema encontram-se alojados nos diversos provedores a exceção das configurações locais torna-se fácil lançar ou recuperar uma réplica que falhou previamente, sendo que a sincronização das réplicas é efetuada a partir dos serviços de clouds.

3.5.4.4 Vantagens e desvantagens do modelo proxy com replicação

Esta variante permite aos utilizadores distribuição de carga pelas várias replicas permitindo que haja mais utilizadores a serem atendidos em simultâneo, nomeadamente em questões de leituras. Como também permite uma tolerância a falhas de servidores, logo permite que se um servidor a executar o middleware encontra-se indisponível, o utilizador possa efetuar as operações a partir de outra réplica na rede.

3.5.4.5 Generalização da arquitetura

Generalizando esta variante podemos ter uma instância do middleware em cada dispositivo de cada utilizador, alternativamente pode-se executar este tipo de aproximação em várias redes locais, sendo elas dispersas geograficamente sem necessidade de encontrarem-se ligadas diretamente entre si. Adicionalmente pode se usar um conjunto de servidores focados para a rede local e um outro conjunto para atender utilizadores a partir da Internet.

3.5.5. Variante MaaS

3.5.5.1 Caracterização desta variante

Para este tipo de utilização, o sistema de *middleware* deverá ser alojado num serviço de *cloud* de computação, que por norma tem custos adicionais de utilização. Como pode ser visto na Figura 3.26 a *cloud* que aloja o serviço de *middleware*, fica responsável por contactar as *clouds* restantes. Será necessário que essa *cloud* seja confiável, pois ira executar operações de cariz sensível sendo que terá que efetuar operações de integridade e encriptação.

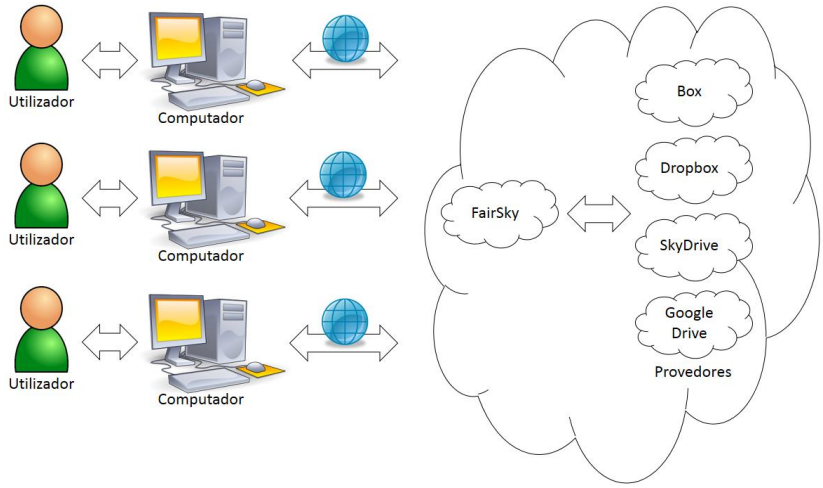

Figura 3.26: Versão MaaS

3.5.5.2 Concretização desta variante

Será necessário uma cloud computacional para este efeito como também confiar na mesma pois ela terá de efetuar as operações criptográficas e oferecer ao utilizador que invoca os dados desencriptados. Assim delega-se a responsabilidade de manutenção e gestão deste middleware mas incorrem outras questões associadas as características impostas pelas soluções. Os clientes irão contactar diretamente a cloud sendo essa agora uma "cloud de armazenamento" a qual ira contactar os provedores de dados.

3.5.5.3 Disponibilidade e Recuperação do Estado de Execução

A disponibilidade desta variante é idêntica a disponibilidade contratada a fornecedor da clouds computacional. Sendo que se optaremos por uma única cloud sem replicação incorremos a disponibilidade imposta por um único ponto de falha.

3.5.5.4 Vantagens e desvantagens do modelo MaaS

Este modelo permite delegar a gestão e manutenção da componente física nomeadamente da infraestrutura a terceiros podendo reduzir potenciais custos e aumentar a disponibilidade do sis-

tema. Também permite aliviar de certa forma a carga na rede, sendo que a comunicação entre o middleware e os diversos provedores de armazenamento é efetuado puramente a partir das cloud que executa o middleware com as clouds reduzindo o uso de recursos. Vantagem interessante é o facto de o serviço encontrar-se alojado numa cloud remota permitindo assim fácil criação de réplicas geograficamente, caso o provedor o permita, fornecendo assim uma solução mais abrangente ao nível de provedor na Internet. Por outro lado delega-se a componente de segurança e privacidade sendo que tem que se confiar na cloud computacional que ira alojar o sistema FairSky, podendo assim esta ser sujeita a ataques maliciosos por parte de 3os, de modo geral um modelo de atacante semelhante ao modelo considerado para os provedores. Questão esta que não é foco desta dissertação.

3.5.5.5 Generalização da arquitetura

Generalizando esta arquitetura podemos usar este sistema como uma versão segura de Dropbox, onde fornecemos a clientes terceiros uma gestão de dados confiável e segura dos seus dados. Permitindo assim usar este sistema como um produto final a outros clientes externos. Será interessante permitir múltiplos utilizadores e gestão de contas neste modelo. Para além de permitir múltiplos utilizadores será então interessante permitir múltiplos perfis para cada utilizador permitindo assim uma solução de associação de perfis.

3.6. Aproximação à implementação do sistema

Tendo em conta a arquitetura inicial de referência e as variantes de generalização anteriormente discutidas, o próximo capítulo será dedicado à implementação do sistema.

O foco principal dessa implementação incidiu numa solução de *middleware* concretizada para armazenamento de ficheiros, seguindo a instanciação da arquitetura apresentada em 3.5.2 (Middleware na variante Cliente Local). Deste modo será abordado em detalhe como a arquitetura de referência foi instanciada para efeitos dessa implementação e consequente avaliação.

4

Implementação

Neste capítulo é descrita a implementação do sistema, com base nas decisões tomadas nos capítulos anteriores relativamente as contribuições desta dissertação. Inicialmente são descritas as tecnologias usadas para o desenvolvimento do protótipo, seguido de uma análise das mesmas como também das ferramentas escolhidas para o desenvolvimento. Serão abordados em maior detalhe o sistema de *profiling* e outros pontos considerados também relevantes.

4.1. Tecnologias utilizadas

O middleware encontra-se implementado em Java. Na conceção e implementação esteve sempre o objetivo de ser o mais fiel possível em relação a arquitetura descrita anteriormente. Deste modo tornasse clara e percetível a separação dos vários componentes, tal como o é na figura de referência. Por outro lado o ambiente Java permite que cada módulo seja facilmente representável por um **package**, como podemos observar na Figura 3.4 tendo em conta a arquitetura implementada.

Foi considerada também a utilização de uma linguagem interpretável, nomeadamente Python, ao contrário de Java que se trata de uma linguagem compilável, pelo facto de haver maior desenvolvimento de componentes e técnicas existentes pela comunidade que suporta Python. O facto de Java permitir que a aplicação seja facilmente testada e executada em ambientes heterogéneos como também em dispositivos móveis foi o fator principal para a sua escolha. Em quanto que em Python, este necessita que o ambiente de execução tenha todas as componentes previamente instaladas e não arquivadas no executável, torna a implementação ligeiramente restrita a ser usada em diversos ambientes.

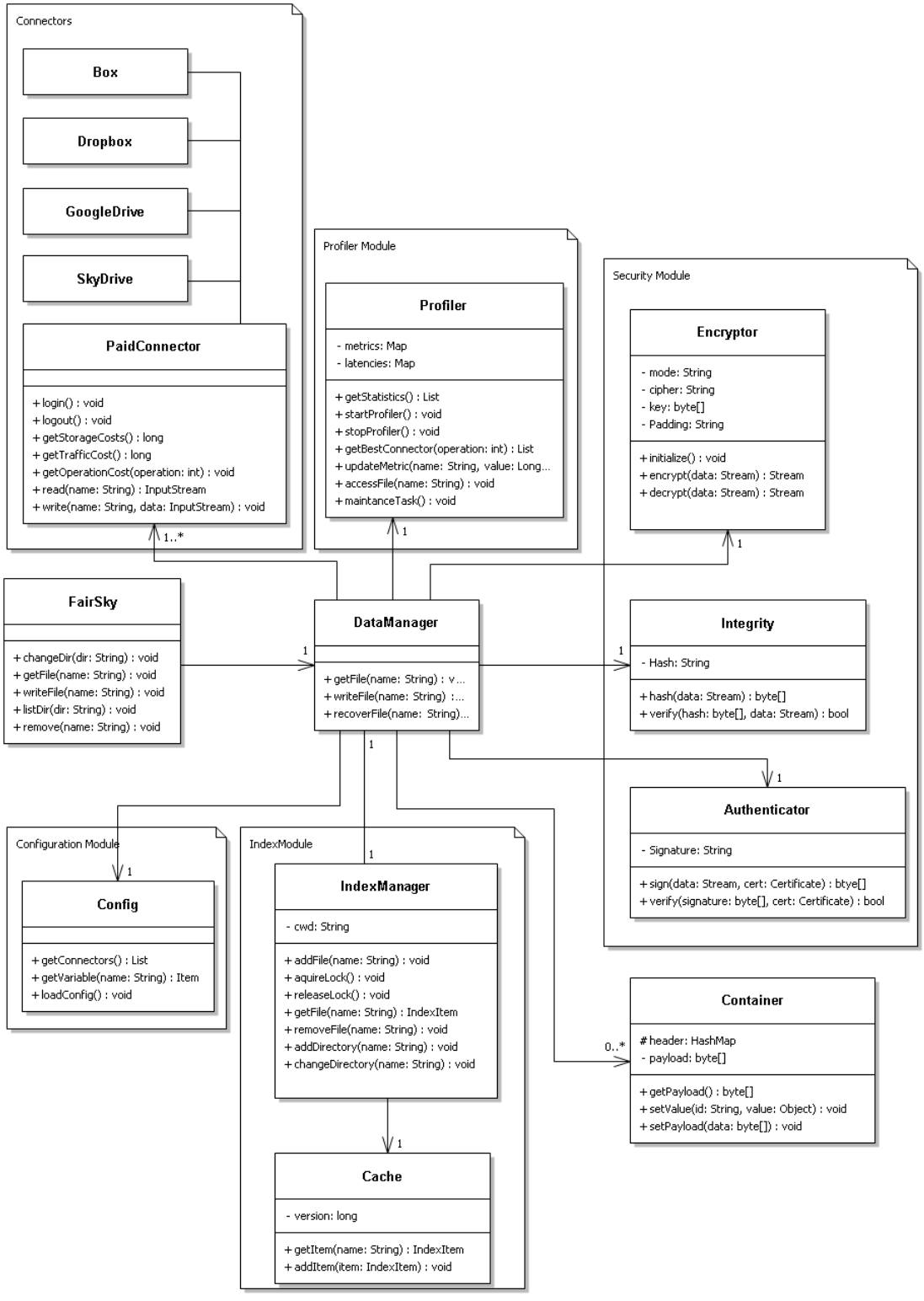

Figura 4.1: Diagrama de classes

4.2. Analise das tecnologias utilizadas

Nesta secção abordam-se as ferramentas utilizadas para o desenvolvimento do protótipo que implementa um middleware de acordo com a arquitetura proposta. São abordados em detalhe o ambiente de desenvolvimento usado como também os diversos componentes desde os componentes as bibliotecas para os conectores em si.

4.2.1. Ferramentas de desenvolvimento

Como referido em cima, o protótipo foi devolvido em Java visto esta ser uma linguagem que fornece garantias de portabilidade e abstração de ambiente de execução, deste modo o protótipo pode ser executado num vasto leque de dispositivos sejam eles computadores em Windows, Linux a *smartphones*.

Para a implementação deste middleware foi utilizado a ferramenta de desenvolvimento Eclipse, que permite simples e eficaz gestão dos recursos e componentes. No que toca a conectores, sendo que cada provedor de armazenamento fornece uma biblioteca própria são usados assim as mesmas consoante o que existe disponível. Por outro lado conectores que não tenham uma biblioteca em Java mas que fornecem unicamente acesso a API via operações REST é usada a biblioteca genérica da Apache que implementa um cliente HTTP. Os componentes de segurança e integridade utilizam algoritmos e bibliotecas estabelecidas e desenvolvidas pela própria Java nomeadamente os provedores criptográficos utilizados na implementação são SunJCE. No que toca a configuração é utilizada a linguagem de notação de objetos JSON armazenada localmente no dispositivo que executa o middleware que permite manter uma coesão e organização das parametrizações dos diversos componentes. Nas secções em seguida iremos abordar em melhor pormenor os componentes em si.

4.2.2. Análise crítica sobre as tecnologias utilizadas

Como abordado anteriormente foi utilizado Java devido a sua portabilidade e utilização de interfaces para abstração de componentes, característica que a linguagem alternativa (Python) não fornece. No que toca aos componentes, optou-se por utilizar a biblioteca SunJCE face a alternativa do provedor bouncy-castle[19] devido ao facto de ambas as implementações fornecem os mesmos algoritmos essenciais nomeadamente no que toca a encriptação de dados: AES, DES e 3DES, os mesmos mecanismos de cálculo de códigos de integridade sendo esses: MD5, SHA1 e SHA-256 e os mesmos mecanismos de caluclo de códigos de autenticidade: MD5withRSA e

[19] http://bouncycastle.org/ (acedido a 20/10/2013)

SHA1withRSA. O uso da biblioteca externa ira causar um acréscimo no tamanho do executável final sem que haja nenhum impacto significativo em nenhum outro aspeto.

4.3. Caracterização da implementação

4.3.1. Profiler

O componente de **profiling** segue o algoritmo proposto em [25], algoritmo esse que determina um valor para cada ficheiro que diz respeito a frequência de acessos ao mesmo. Com base nesse valor associado podemos decidir se um ficheiro devera ser ou não considerado como um ficheiro de pouco uso, permitindo ao **profiler** recolher métricas a escolher conectores cujo plano de custos se enquadre melhor no perfil de uso do ficheiro. Como abordado no capítulo anterior o **profiler** suporta métricas de custos e métricas de latências. Foi criado assim um componente de perfis de custos permitindo ao utilizador estabelecer critérios de escolha nomeadamente com base nos custos de cada operação e na latência. Assim o cliente tem maneira de decidir com base em que critérios são feitas as escolhas do **profiler**. Em seguida apresentamos os métodos da interface de execução de critérios. Como o sistema FairSky foi implementado em Java, as políticas são ficheiros que contem métodos, ficheiros esses escritos em Javascript assim é simples e eficaz executar código dinâmico por parte do middleware para calcular os custos.

```
public interface RunScript
{
	public static double runFast(double latency, double store_cost,
		double network_cost, double operation_cost);

	public static double runArchive(double latency, double store_cost,
		double network_cost, double operation_cost);

	public static double runCheapRead(double latency, double store_cost,
		double network_cost, double operation_cost);

	public static double runCheapWrite(double latency, double store_cost,
		double network_cost, double operation_cost);

}
```

Listagem 4.1: Interface de processamento de políticas

A latência usada é calculada com base em lista de amostras, amostras essas recolhidas durante a execução do FairSky. Essa lista tem um tamanho definido, e com base nesses elementos é efetuada uma média aritmética dos elementos e usado esse valor. Para a implementação dessa fila usou-se um Buffer Circular sendo que são descartados sempre os valores mais antigos. Este *buffer* permite alguma tolerância a situações onde a latência é sujeita a particularidades da rede onde os dados são recolhidos nomeadamente alguma perda de pacotes ou saturação da mesma.

Situações essas que são amortizadas pelo uso da média. Caso se usasse uma única amostra por conector correr-se-ia o risco da mesma inibir o uso de um provedor que monetariamente foi sujeito a estes comportamentos da rede.

O pseudocódigo seguinte ilustra o processamento da implementação do FairSky referente ao processamento do Profiler durante a operação de escrita de um objeto nos diversos provedores. É criado um mapa e nele são colocados os custos e o respetivo conector. Extrai-se de cada conector as métricas necessárias para efetuar os cálculos nomeadamente: a latência, o custo de invocar o comando PUT, o custo de transferência do conteúdo e também o custo de armazenar o objeto. Em seguida tendo as métricas todas, invoca-se a validação e cálculo da política estabelecida. São passados em argumentos esses valores e é devolvido o custo com base na política. Insere-se no mapa usando o custo como chave e valor o identificador do conector. No fim de percorrer e recolher os custos para todos os conectores, ordena-se o mapa tendo no início do mesmo o melhor conector a usar.

Operação 9 Operação de getConnectorsWrite

```
1:  procedure GETCONNECTORSWRITE(metaData)
2:      size ← getFileSize(metaData)
3:      map = { }
4:      allConnectors ← getAllConnectors()
5:      for connector in allConnectors do
6:          latency ← getLatency(connector)
7:          opCost ← getOperationCost(connector, PUT)
8:          trafficCost ← getTrafficCost(connector, size)
9:          storageCost ← getStorageCost(connector, size)
10:         cost ← processPolicy(latency, opCost, trafficCost, storageCost)
11:         map.add(cost, connector)
12:     end for
13:     sortByKey(map)
14:     return map
15: end procedure
```

Listagem 4.2: Pseudocódigo de operação do *profiler*

4.3.2. Conectores

O uso de Interfaces do Java permite que possa haver abstração entre os diversos módulos e especialmente entre os conectores permitindo que o sistema seja modular e facilmente mantido. Assim, por exemplo, se desejamos adicionar conectores adicionais temos que respeitar uma das 2 interfaces disponibilizadas: *Connector* ou a sua extensão *PaidConnector* como podemos ver na Listagem 4.3. *Connector* afeta provedores que não tenham tarifários impostos nomeadamente planos de serviços gratuitos ou armazenamento local, enquanto que PaidConnector incorpora adicionalmente métodos que permite calcular os custos e o respetivo tarifário do provedor.

```
public abstract class Connector
{

	public abstract void init(ProviderConfig pc);
	public abstract void write(Container container);
	public abstract Container readContainer(String item);
	public abstract void delete(String item);
	public abstract void search(String item);
	public abstract void login();
	public abstract void logout();
	public abstract String getConnectorIdentifier();
	public abstract ProviderConfig getConfig();
	public abstract void write(String filename, InputStream fis);
	public abstract void write(String filename, File f);
	public abstract InputStream readStream(String item);
	public String getEncodedName(String name, String mac);
}


public abstract class PaidConnector extends Connector
{
	public abstract double getStoreDataCost(long total_size);
	public abstract double getDataTrafficCost(long total_size);
	public abstract double getReadOperationCost(long num_operations);
	public abstract double getWriteOperationCost(long num_operations);
}
```

Listagem 4.3: Código interface dos conectores

Um dos componentes que provou ser mais complicado de implementar foi a integração do *profiler* com os restantes componentes. Nomeadamente optou-se por delegar os cálculos dos custos para os módulos de cada conector, deste modo permite-se que o sistema suporte múltiplos heterogéneos tarifários com diferentes tipos de variáveis.

4.3.3. Gestor de Dados

A Listagem 4.4 apresenta o algoritmo correspondente a escrita de um ficheiro no middleware, com foco particular nas operações e interações do *profiler*. Como podemos observar o *profiler* é consultado antes e depois de cada operação, mais concretamente antes para poder filtrar e decidir quais os conjuntos de provedores adequados para uma dada operação enquanto que depois da operação atualiza-se a informação existente no *profiler* sobre uma dada cloud, nomeadamente para confirmar que a operação foi concluída, a modos que o *profiler* tenha informação atualizada para consultas futuras.

```
public void send_file(String file)
{
	sign(file, user_private_key);
	compress(file);
	encrypt(file);
	fragment_file(file);
	for each fragment in fragments
	{
		hash(file);
		connectors = profiler.get_best_connectors(info);
		for each connector in
		{
			connector.upload(file_name);
		}
	}
	profiler.update(info);
	index.add(file, connectors, fragments);
}
```

Listagem 4.4: Pseudocódigo de escrita de um ficheiro

```
public void get_file(String file_name)
{
	info = index.get_info(file_name);
	verify_is_not_folder(info);
	for each fragment in get_fragments(info)
	{
		connector = profiler.get_best_connector(info);
		container = connector.download(file_name);
		verify_integrity(container);
		if( error )
		{
			byzantine_recover(file_name);
		}
	}
	data = merge_all_fragments();
	decrypt(data);
	decompress(data);
	authenticate(data);
	output.write(data);
	profiler.update(info);
}
```

Listagem 4.5: Pseudocódigo de leitura de um ficheiro

Como pode-se ver na Listagem 4.5 o código de uma leitura de um ficheiro segue o mesmo estilo que uma escrita sendo que as únicas diferenças são as informações armazenadas no fim no profiler e o modo de operação dos restantes componentes. Também foi implementada uma *Thread* para ir periodicamente atualizar os valores associados a cada ficheiro, como foi apresentada a formula na secção anterior. Alternativamente o profiler, como mencionado ao longo deste documento, pode operar em modo de latências. Para efeito foi desenvolvido uma

estrutura especial que implementa Listas em Java, listas essas que dado um limite de elementos quando se adicionam novos a eles verifica se pode descartar existentes sendo que com a nova adição não ultrapassar o limite imposto. Assim com esta lista limitada, ao longo da execução nomeadamente na fase de atualização de informação, recolhe-se os tempos de execução e tempos de respostas e esses valores são adicionados as listas restritas de cada um dos conectores. Ambos os modos funcionam em paralelo, sendo que o utilizador pode, recorrendo ao ficheiro de configuração, escolher com que modo pretende que a escolha de clouds candidatas devera ser efetuada.

4.3.4. Módulo de Segurança e Integridade

Os módulos de segurança, integridade e autenticidade são tratados pelos seus respetivos módulos que se encontram no *package* **Security**. As operações destes módulos são operações básicas sobre *Streams* de dados, sendo que só é necessário para a segurança: encriptar e desencriptar, para a integridade calcular o código de integridade e comparar uma *Stream* face a um código e a autenticidade funciona de modo semelhante a integridade visto adicionalmente necessitar de uma chave privada para a geração dos códigos ou a chave pública para a verificação do conteúdo.

```
public class Encryptor
{
	public byte[] update(byte[] in);

	public byte[] encrypt(byte[] in);

	public byte[] decrypt(byte[] in);

	public CipherInputStream getInputStream(InputStream is);

	public CipherOutputStream getOutputStream(OutputStream is);

	public static byte[] encryptOrDecryptWithPBE(byte[], char[]);

	public boolean usesIV();

	public byte[] getIV();

	public void setIV(byte[] newIV);

	public void setKey(byte[] newKey);

	public byte[] generateIV();

}
```

Listagem 4.6: Métodos disponíveis pelo módulo de encriptação

```
public class Integrity
{
	public byte[] hash(FileInputStream input);

	public boolean verifyHash(FileInputStream input, byte[] hash);
}
```

Listagem 4.7: Métodos disponíveis pelo módulo de integridade

4.3.5. Índice

O índice, não sendo ele foco principal desta dissertação, foi implementado com duas versões a vista, uma interna e uma externa, ambas a respeitarem a interface apresentada na Listagem 4.8.

A interface interna foi implementada puramente recorrendo ao próprio FairSky, nomeadamente utiliza o armazenamento de objetos como repositório do índice. O objetivo desta vertente é, para além de respeitar os critérios do próprio middleware nomeadamente: minimizar os acessos necessários, ser tolerante a falhas e os requisitos principais da própria dissertação. Assim deste modo, para minimizar os acessos o índice esta organizado em listas, sendo que cada ficheiro ou diretoria é representado por um ficheiro que contem ou informação associada ao ficheiro sendo essa informação, identificação dos fragmentos, localização dos fragmentos e tamanho de ficheiro ou então uma lista de identificadores de ficheiros existentes caso se trate de uma diretoria. Assim com o funcionamento usual do middleware, atinge-se uma redução de acessos aos índices nomeadamente não é necessário obter grandes volumes de listas para ficheiros que não são necessariamente acedidos. Por outro lado foi implementado um componente auxiliar de *Cache*. Este componente entra em funcionamento no início de cada pesquisa, sendo que é mantida assim uma listagem local da informação do índice remoto. O objetivo deste componente é minimizar os acessos aos diversos provedores para obter a informação desejada sendo que pode-se também fornecer informação mais rápida sobre os conteúdos caso eles se encontrem alojados na *Cache* local, sendo essa cloud atualizada após cada operação com os dados da mesma. Na Listagem 4.9 podemos observar como se comporta o sistema de índice durante uma operação de leitura de um ficheiro existente na cloud.

A versão externa, a qual foi desenvolvida a ser usado um índice recorrendo ao modelo Paxos permite declarar um URL de um serviço REST para operação do índice. Foco desta versão foi a utilização do OpenReplica[20] para gerir e operar o índice. Assim foi implementado um componente que permite dado um URL efetuar todas as operações especificadas na interface mas sobre esta ferramenta externa. Visto o OpenReplica ser feito em Python, obrigou a criação

[20] http://openreplica.org/ (acedido a 21/10/2013)

de um componente adicional em Python capaz de receber os pedidos REST do middleware e interagir com os objetos do OpenReplica.

```
public interface IndexManager extends Serializable
{
	public String addFile(String name, int fragments);

	public void addFragment(String file_name, String fragment_name, int
		fragment_id, String connector_id, long size);

	public void addDirectory(String name);

	public void createItem(String name, boolean isDir, int fragments);

	public LinkedList<String> listDir();

	public Item getFile(String name);

	public void delete(String name);

	public LinkedList<String> search(String search);

	public void setFile(String key, IndexFile inf)

	public String currentPath();
}
```

Listagem 4.8: Interface do gestor de Índice

```
public Item getFile(String name)
{
	if (cache.is_valid())
	{
		if (cache.contains(name))
			return cache.get(name);
	}
	else
	{
		cache.clear();
	}
	//if cache is invalid or cache does not have item
	//get the file via the middleware's get_file()
	index_item_name = convert_name(name);
	index_item = datamanager.get_file(index_item_name);
	cache.add(index_item);
	return index_item;

}
```

Listagem 4.9: Pseudocódigo de consulta de índice

4.3.6. Contentor

O modelo de dados foi implementado seguindo uma estrutura própria do Java, nomeadamente *HashMap*, cuja chave é um identificador de texto e o valor é um Objeto genérico de Java. Ao fazer esta aproximação é possível armazenar nos cabeçalhos dos ficheiros qualquer e modelo de dados de Java permitindo a abstração para cada módulo. Mais concretamente os códigos de integridade e autenticidade são *byte*[], sendo que esses mesmos são armazenados no mapa sem qualquer operação adicional por parte dos módulos ou do tratamento de dados. Importante é salientar que devera haver uma coesão nas chaves do mapa como também cada componente devera saber que tipo de dados armazena no mesmo.

4.4. Métricas de implementação

Como mencionado anteriormente, este middleware foi principalmente implementado em Java mas também foi desenvolvido parcialmente em paralelo em Python. Mais concretamente embora não tenha sido implementado e testado na sua totalidade o middleware em Python, foi possível extrair algumas métricas de implementação para comparar com Java. Particularmente existe maior oferta de bibliotecas por parte da comunidade referente a algoritmos e procedimentos de diversos componentes. Mais concretamente, na questão do desenvolvimento do conector da SkyDrive, em Java teve que ser desenvolvido desde raiz um conector REST que funcionasse perfeitamente com a API providenciada, e mesmo assim existem diversas dificuldades com a vertente visto haverem certas implicações com a autenticação OAuth 2 que a API usa, enquanto que em Python existe um modulo que age já como cliente para essa API, logo já neste conector podemos observar que o esforço necessário por parte do programador. Em modo semelhante se observa a implementação do conector para a Google Drive onde embora a biblioteca em Java esteja melhor implementada, a secção que trata da autenticação em OAuth ocupa bastantes linhas de código face a implementação em Python. Em relação aos restantes módulos, dos que foram implementados, temos na Tabela 4.1 uma comparação entre linhas de código necessárias para implementar as mesmas funcionalidades, entre Java e Python.

Camada na arquitetura	Componente	Java	Python
Tier 1	Middleware API	672	Não Implementado
Tier 2	Profiler	734	Não Implementado
	SecurityManager	317	171
	IndexManager	776	Não Implementado
	DataManager	461	378
Tier 3	Conector SkyDrive	301	68
	Conector Google Drive	320	64
	Conector Dropbox	260	63
	Conector Box	318	61

Tabela 4.1: Linhas de código por módulo para cada linguagem

Como podemos observar na tabela anterior, existe uma pequena diferença entre a implementação em Java face a implementação em Python. Maior discrepância dos valores notam-se nos conectores, isto deve-se maioritariamente ao facto de as bibliotecas utilizadas em Java não terem implementadas as funcionalidades que as equivalentes em Python tem, sendo que o objetivo em Java é serem abstratas e genéricas enquanto que em Python tem como foco principal facilitar a vida do programador. Por exemplo o conector da Google Drive em Python implementa por si um módulo que trata internamente a gestão da autenticação por OAuth 2 lançando o navegador de Internet com o pedido ao utilizador para validação de acessos, enquanto que em Java essa funcionalidade teve que ser implementada do lado da tese. Esta condicionante verifica-se também nos restantes conectores a exceção da Dropbox, tornando assim a implementação em Python mais reduzida e com menos esforço por parte do programador.

4.5. Aspetos em aberto

Tendo em vista os componentes que foram implementados e as suas particularidades podemos seguramente concluir que os aspetos em aberto desta implementação encontram-se a volta de alguns componentes.

Um dos aspetos pouco explorados desta implementação é a gestão e organização do índice. Embora a mesma tenha sido feita com conta a redução de acessos, deveram existir outras ferramentas e soluções que permitam fornecer uma gestão mais flexível e simples do mesmo, garantindo informação fiável e segura dos objetos armazenados. Alternativamente para esta questão poder-se-ia usar informação armazenada remotamente já processada e cifrada, sendo que o índice poderá ser guardado em pleno texto, a fins de reduzir o processamento por parte do middleware, tornado as operações sobre o índice mais diretas.

Aspeto importante para desenvolvimento futuro será adicionar mais conectores para suportar melhor o vasto leque de provedores existentes hoje em dia no mercado, tornando assim possível melhorar o modelo de falhas para ter em consideração mais servidores bizantinos. Como também melhorar os conectores existentes, nomeadamente o conector da SkyDrive tem cer-

tas limitações na sua implementação devido ao facto de utilizar diretamente a API do provedor e não ter uma biblioteca robusta e com mais funcionalidades, deste modo foram deixados por fora a autenticação recorrendo a carregamento de um navegador de Internet para concluir o protocolo OAuth, sendo que o utilizador deve especificar manualmente as chaves de credenciais no ficheiro de configuração, em contraste com a Google Drive e Box que efetuam essa autenticação com o consentimento do utilizador.

Embora a implementação do Profiler aparenta ser "boa", como se ira verificar mais a frente, poderá ser interessante melhorar esta componente a fins de permitir modelos de custos parametrizados nomeadamente custos associados a diretorias especificas caso o provedor forneça tal opção, ou então seria interessante também extrair as métricas a partir dos provedores que oferecem as mesmas e usadas para se compararem com as métricas locais, nomeadamente em questões de espaço ocupado e de datas de acessos aos diversos ficheiros. Deste modo ao ter mais informação relacionada com os acessos e configurações da conta do utilizador será interessante expandir a configurabilidade e capacidade do profiler em si, podendo fornecer melhores sistemas de profiling que sejam mais baratos para o utilizador. Adicionalmente seria interessante neste tópico desenvolver outros tipos de operação do mesmo ou então misturar modos de operação, especificamente misturar o modo de latências com o modo de redução de custos onde o cliente final poderá estar disposto a pagar algo mais para ter acesso mais rápido aos ficheiro mas não quer obrigatoriamente ir pela solução mais cara. Sendo assim estabelecida uma relação custo/latência para o profiler.

4.6. Aproximação a avaliação da implementação

Tendo abordado a arquitetura previamente e neste capítulo apresentado as diversas interfaces e metodologias particulares da apresentação, no seguinte capítulo vamos testar, avaliar e analisar a implementação deste sistema. Foco principal da implementação é o *profiler*, o qual será testado e avaliado enquanto que serão também abordadas as restantes métricas produzidas a partir do impacto dos restantes componentes e da sua parametrização.

5

Avaliação

Neste capítulo é feita a avaliação do sistema implementado, no que diz respeito ao seu desempenho e garantias de segurança e redução de custos ou latências que oferece. Inicialmente, é efetuada uma comparação da utilização do *profiler*, focando nos modos particulares de operação do mesmo, face a otimização de latências, redução de custo de funcionamento e também a migração de dados.

A avaliação e validação do sistema procura dar respostas às seguintes questões:

- O impacto de latência de escritas e leituras com base na utilização dos serviços do middleware FairSky é significativo quando comparado com a utilização direta de uma cloud?
- O facto de o middleware FaiSky usar múltiplas nuvens com as vantagens que daí se podem tirar?
- O facto de o middleware FairSky migrar os dados de uma cloud para outra consoante os critérios, reduz os custos finais ao utilizador?
- É vantajoso migrar os dados entre as clouds se pretendemos reduzir as latências?

5.1. Ambiente e métricas de avaliação

O ambiente de operacionalização e avaliação do middleware e dos testes apresentados ao longo deste capítulo está resumido na seguinte tabela.

<table>
<tr><td colspan="2">Ambiente computacional</td></tr>
<tr><td>Hardware:
Tipo de arquitetura: 64 bit
Processador: Intel Core i5 @ 2.27 GHz
Número de cores físicos: 2
Memoria Física: 4 GB
Disco Rígido: 128GB, SSD, interface SATA</td><td>Software
Sistema operativo: Microsoft Windows 8 Professional
Java Runtime Environment 6 . (Oracle Standard Distribution)
Suporte JCE: provedores criptográficos (Sun JCE)
Pyton 2.7
OpenReplica</td></tr>
<tr><td colspan="2">Comunicações</td></tr>
<tr><td colspan="2">Ligação Middleware-Internet: ADSL
Largura de banda de referência: 4096 Kbit/s downstream - 1024 Kbit/s upstream.</td></tr>
<tr><td colspan="2">Cloud Storage Providers</td></tr>
<tr><td>Testes de latências foram
Google Drive
SkyDrive
Dropbox
Box</td><td>Testes de custos foram:
Amazon S3
Microsoft Azure Cloud Storage
Google Cloud Storage
Rackspace Cloud Files</td></tr>
</table>

Tabela 5.1: Ambiente Computacional

O cliente que utiliza o middleware executa na mesma máquina em que este se encontra instalado e em execução, sendo que todas as operações serão executadas localmente, nomeadamente entre cliente e middleware. Enquanto que o middleware terá que comunicar a partir do computador com cada uma das clouds acima mencionadas de acordo com o teste a apresentar. Esta instalação corresponde à instanciação da visão da arquitetura inicialmente apresentada no capítulo 3, secção 3.5.2.

5.2. Impacto da solução FairSky

O objetivo deste conjunto de testes iniciais, é comparar o impacto dos diversos componentes do middleware na sua operação. Sendo que em primeira fase testam-se diretamente as clouds, re-

correndo aos acessos diretos as suas APIs, para poder extrair informação sobre o desempenho das mesmas independentemente do middleware. Em seguida ira avaliar-se o *trade-off* de utilização do middleware recorrendo a melhor cloud escolhida no primeiro conjunto de testes face a utilizando da mesma mas sem o middleware. Por fim será abordado o impacto da solução recorrendo a múltiplas clouds face a utilização de uma única só, como objetivo desta questão final será abordar a utilização de replicação e o seu impacto no sistema.

5.2.1. Avaliação comparativa da latência de operações com diversas nuvens de armazenamento

Para este teste utilizamos diretamente as APIs de cada provedor, sendo efetuada uma variação de tamanhos de ficheiros. Para cada tamanho, efetuam-se 10 testes e são apresentados em seguida os seus respetivos valores para operações de leitura ou de escrita.

A Tabela 5.2 apresenta valores da operação de leitura de diversos ficheiros em segundos. Como se pode observar a Dropbox apresenta valores de obtenção de dados relativamente baixos em comparação com os outros provedores testados para ficheiros de 10Kb, 100Kb e 1 Mb enquanto que para ficheiros de 10Mb, verifica-se que a Google Drive tem os menores tempos, importante é destacar o facto que para 1 Mb, a Google Drive encontra-se bastante perto dos valores da Dropbox. Assim para as operações de leitura destacamos e consideramos a Dropbox como a "melhor" cloud. Claramente com o aumento do tamanho de ficheiro, o tempo de transmissão aumenta também.

Tamanhos	SkyDrive (seg)	Dropbox (seg)	Box (seg)	Google Drive (seg)
0.01 MB	5.530 ±0.139	0.719 ±0.253	4.964 ±0.393	1.167 ±0.256
0.1 MB	6.084 ±0.204	0.998 ±0.371	5.562 ±0.627	1.408 ±0.312
1 MB	7.776 ±0.161	2.765 ±0.472	7.656 ±0.254	3.029 ±0.246
10 MB	25.723 ±0.457	20.184 ±2.278	23.478 ±0.937	19.310 ±0.375

Tabela 5.2: Tempos (em segundos) de operação de leitura direta nas clouds

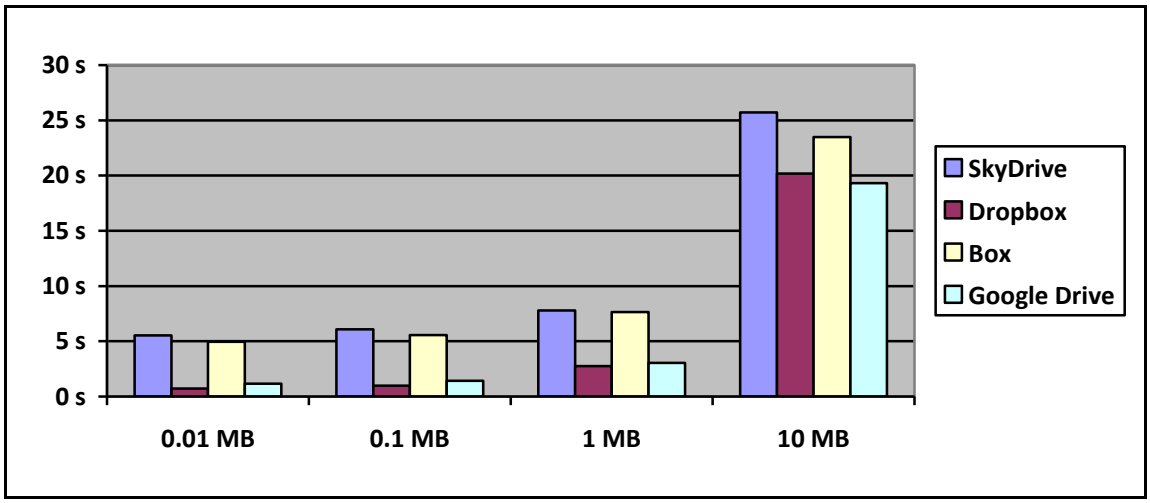

Figura 5.1: Gráfico correspondente aos valores da Tabela 5.2

Em seguida efetuamos o mesmo teste para escritas de ficheiros de tamanho variado. Como podemos observar naTabela 5.3, os valores são superiores as leituras para grandes ficheiros, isto deve-se ao facto que a velocidade de *upstream* é inferior a velocidade de *downstream*. Nestes resultados podemos observar que a Dropbox continua a ser a mais rápida para ficheiros de 10Kb, 100Kb e 1MB, enquanto que para ficheiros maiores (10Mb) a Google Drive apresenta os tempos mais reduzidos em comparação com as restantes. Assim para as operações de escrita destacamos como "melhor" a Dropbox dado o facto de se comportar melhor em 3 dos 4 tamanhos considerados.

Tamanhos	SkyDrive (seg)	Dropbox (seg)	Box (seg)	Google Drive (seg)
0.01 MB	3.747 ±0.173	0.786 ±0.527	3.904 ±0.428	7.515 ±3.708
0.1 MB	4.925 ±0.286	1.912 ±0.438	4.981 ±0.533	7.334 ±0.259
1 MB	15.056 ±0.758	12.129 ±0.781	15.007 ±0.931	14.407 ±0.767
10 MB	111.705 ±2.425	108.366 ±0.608	109.714 ±1.665	106.483 ±0.671

Tabela 5.3: Tempos (em segundos) de operação de escrita direta nas clouds

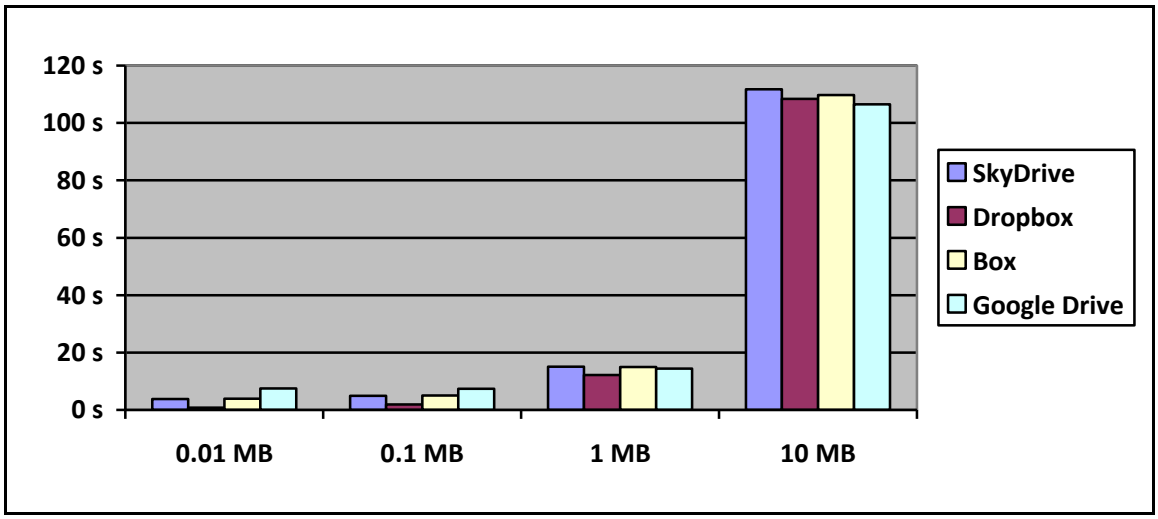

Figura 5.2: Gráfico correspondente aos valores da Tabela 5.3

5.2.2. Comparação face à melhor nuvem

Tendo em vista os resultados obtidos da comparação direta entre as clouds, vamos agora comparar os resultados obtidos para o mesmo conjunto de ficheiros mas comparando a "melhor" cloud escolhida anteriormente face ao sistema middleware a ser executado na mesma cloud. Na Tabela 5.4 encontram-se as médias de 10 medições onde podemos observar que o middleware FairSky apresenta, como esperado, valores de leituras de ficheiros ligeiramente superiores as equivalentes de uso direto da "melhor" cloud. Isto deve-se ao facto que o sistema fornece garantias de segurança confidencialidade. Logo o acréscimo de tempos deve-se ao processamento dos diversos componentes. Observa-se que esses tempos não são de grande significância face as propriedades que se oferecem, enquanto que o uso de uma cloud diretamente não oferece garantias de confidencialidade e privacidade.

Tamanho	Dropbox (seg)	FairSky (seg)
0.01 MB	0.719 ±0.253	0.789 ±0.163
0.1 MB	0.998 ±0.371	1.031 ±0.287
1 MB	2.765 ±0.472	3.028 ±0.332
10 MB	20.184 ±1.278	22.620 ±0.742

Tabela 5.4: Tempos (em segundos) de leitura de ficheiros

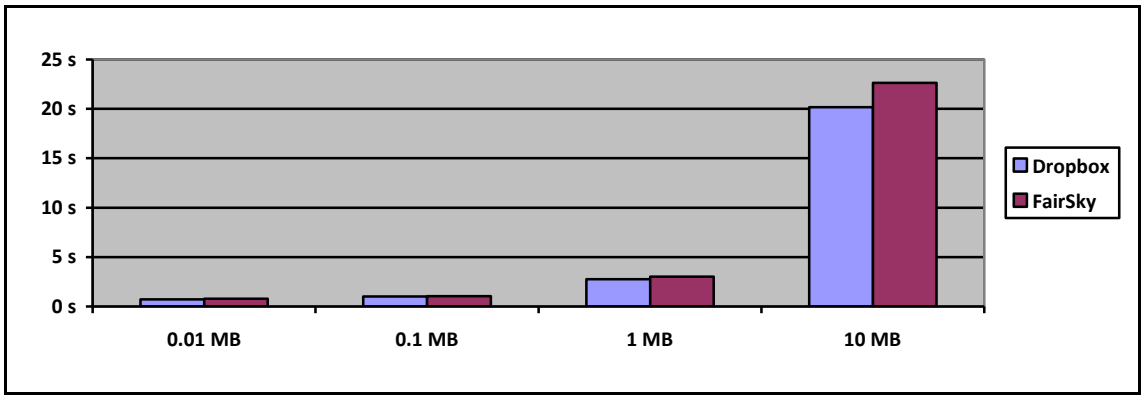

Figura 5.3: Gráfico correspondente aos valores da Tabela 5.4

O mesmo podemos observar nos testes de escritas de ficheiros, os resultados dos quais apresentados em Tabela 5.5, onde o acréscimo de tempos introduzido é pequeno face as garantias que são oferecidas pelo middleware. A semelhança das leituras podemos observar que quanto maior o ficheiro, mais tempo é necessário por parte do middleware para o processar, nomeadamente calcular os códigos de integridade e autenticidade como também efetuar as operações de criptografia sobre os ficheiros. Mas grande parte do tempo de operação deve-se aos grandes tempos de transmissão de dados, transmissão essa que esta limitada a capacidade de envio do dispositivo onde o middleware executa. De modo idêntico a Tabela 5.5 apresenta as médias obtidas de 10 execuções consecutivas para cada um dos tamanhos e modos apresentado.

Tamanho	Dropbox (seg)	FairSky (seg)
0.01 MB	0.786 ±0.527	0.831 ±0.144
0.1 MB	1.912 ±0.438	2.163 ±0.104
1 MB	12.129 ±0.781	13.967 ±0.251
10 MB	108.366 ±0.608	113.805 ±0.483

Tabela 5.5: Tempos (em segundos) de escrita de ficheiros

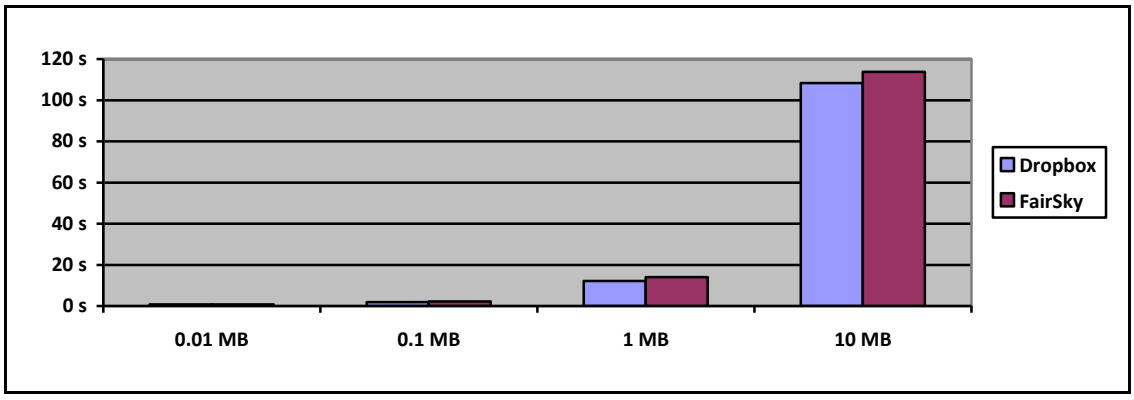

Figura 5.4: Gráfico correspondente aos valores da Tabela 5.5

Concluímos assim que para este teste respondemos a pergunta apresentamos inicialmente:

- O impacto de latência de escritas e leituras com base na utilização dos serviços do middleware FairSky é significativo quando comparado com a utilização direta de uma cloud?

A qual podemos responder que a diferença tanto para escrita como leitura é pequena face ao ganho de segurança integridade e autenticidade dos dados colocados na cloud.

5.2.3. Comparação de uso de múltiplas nuvens face à melhor nuvem

Para este teste consideramos os valores obtidos anteriormente, utilizando unicamente a Dropbox como provedor para o middleware, face a utilização de todos os provedores. Nomeadamente neste teste avaliamos a vantagem e o impacto de replicação de objetos por múltiplas clouds. Sendo que iremos demonstrar o tempo necessário acréscimo para garantir recuperabilidade e fiabilidade do conteúdo colocado nele. Assim para o mesmo conjunto de ficheiros, recorrendo a 10 amostras por cada operação por cada ficheiro temos os seguintes resultados.

Na Tabela 5.6 podemos observar a comparação de tempos de execução de utilização de uma única cloud face a utilização de replicação por todas as clouds. Como era de esperar, a replicação por todas tem valores muito superiores as de uma cloud, isto deve-se ao facto da operação de leitura efetuar-se numa base aleatória, sendo que é escolhido o provedor a obter um dado ficheiro puramente aleatório. Isto torna o middleware no pior caso sujeito a latências da pior cloud usada.

Tamanho	FairSky com uma cloud (seg)	FairSky com múltiplas clouds (seg)
0.01 MB	0.789 ±0.163	2.988 ±2.340
0.1 MB	1.031 ±0.287	3.398 ±2.403
1 MB	3.028 ±0.332	5.373 ±2.491
10 MB	22.620 ±0.742	24.742 ±2.725

Tabela 5.6: Tempos (em segundos) de leitura de ficheiros

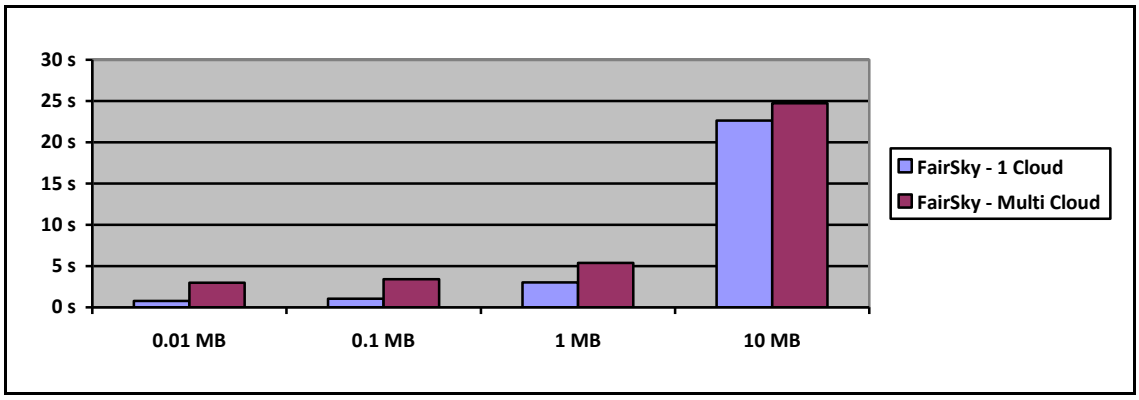

Figura 5.5: Gráfico correspondente aos valores da Tabela 5.6

Como era de esperar, na Tabela 5.7, os tempos de escrita numa única cloud são sempre inferiores que escrever em todas as clouds. Como nesta situação é obrigatório escrever em 4 clouds, o uso de múltiplas clouds, esta sempre sujeito aos tempos de execução de todas as clouds usadas. Como se pode observar os tempos de escrita são mais elevados que os tempos de leitura, não só devido ao facto de limitações de canal de transmissão mas devido ao facto que a leitura no pior caso acede a pior cloud disponível, enquanto que o caso esperado para a replicação por todas as clouds é o pior caso, nomeadamente enviar para todas as clouds.

Tamanho	FairSky com uma cloud (seg)	FairSky com múltiplas clouds (seg)
0.01 MB	0.831 ±0.144	10.701 ±0.164
0.1 MB	2.163 ±0.104	15.387 ±0.194
1 MB	13.967 ±0.251	58.529 ±0.740
10 MB	113.805 ±0.483	456.476 ±1.377

Tabela 5.7: Tempos (em segundos) de escrita de ficheiros

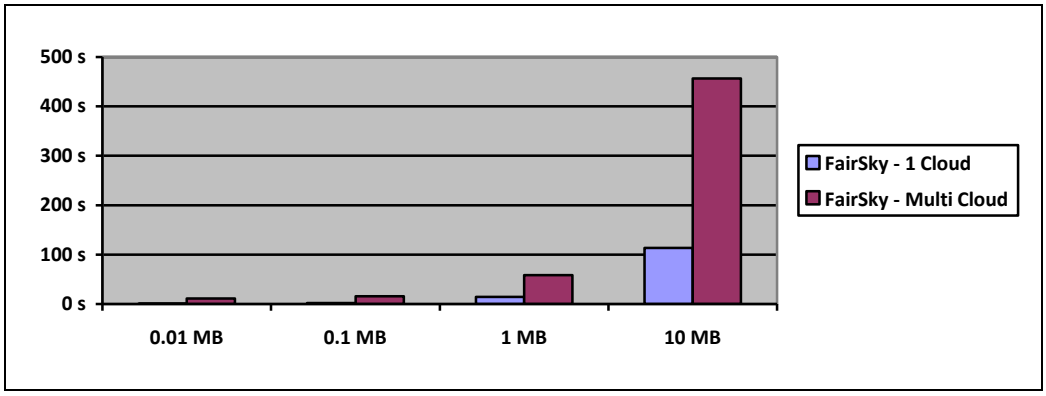

Figura 5.6: Gráfico correspondente aos valores da Tabela 5.7

5.3. Impacto dos componentes da arquitetura FairSky

Nesta secção abordamos e avaliamos o impacto dos diversos componentes do sistema no seu desempenho global variando os diversos modos de configuração dos mesmos. Aborda-se em primeira fase o componente de segurança, seguido da integridade dos dados e por fim a autenticidade do conteúdo. Importante fator a destacar neste conjunto de teste é as operações de criptográfica, integridade e autenticidade serem efetuadas sempre antes de cada operação. Deste modo a utilização de uma ou múltiplas clouds não tem impacto sobre estes componentes, sendo os seus tempos iguais para qualquer número de clouds a usar pelo FairSky. Assim os seguintes testes limitam-se a comparar entre si as diversas configurações dos mesmos.

5.3.1. Criptografia

Visto um dos focos deste sistema é a segurança e confidencialidade dos dados, foi efetuado um estudo sobre quatro modos principais de cifra de dados. Cifras essas disponíveis pelo provedor SunJCE. Deste modo prossegue-se a comparação do AES nas suas 3 variantes de tamanhos de chaves como também a cifra Triple DES, recorrendo a tempos de encriptação de diversos tamanhos de ficheiros.

Tamanho	AES 128 (seg)	AES 192 (seg)	AES 256 (seg)	Triple DES (seg)
0.01 MB	0.016	0.015	0.015	0.043
0.1 MB	0.033	0.035	0.037	0.043
1 MB	0.044	0.047	0.049	0.162
10 MB	0.399	0.409	0.413	1.502

Tabela 5.8: Tempos de execução (em segundos) de encriptação

Como podemos observar na Tabela 5.8, o algoritmo AES é bastante rápido em comparação com o Triple DES. Entre as diferentes variantes do algoritmo AES os valores observados tem valores muito próximo. Assim para a utilização neste sistema optou-se pela variante AES 256. Variante essa que mostra ser mais adequada devido ao ganho de segurança face ao tempo de processamento acréscimo necessário. Verifica-se que esse *trade-off* justifica-se para ficheiros superiores a 1 MB.

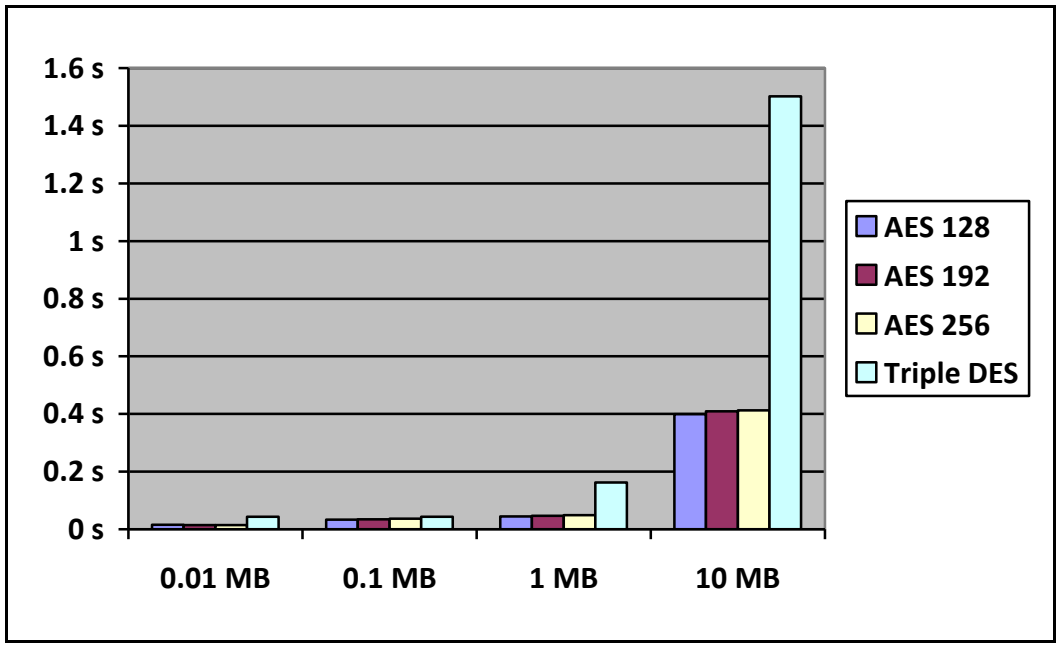

Figura 5.7: Gráfico correspondente aos valores da Tabela 5.8

5.3.2. Integridade

Nesta secção abordamos o impacto do componente de integridade. Componente este que calcula um código de integridade associado a cada ficheiro. Para este teste consideramos, similarmente um conjunto de algoritmos, nomeadamente SHA1, SHA-256 e MD5, variando para cada um deles os ficheiros. Após a conclusão dos testes temos a seguinte tabela que resulta da média de 10 amostras retiradas da operação de escrita

Tamanho	MD5 (seg)	SHA 1 (seg)	SHA 256 (seg)
0.01 MB	0.001	0.001	0.005
0.1 MB	0.007	0.006	0.010
1 MB	0.011	0.019	0.021
10 MB	0.099	0.144	0.165

Tabela 5.9: Tempos (em segundos) da operação de integridade

Como podemos observar na Tabela 5.9, o algoritmo MD5 apresenta os tempos mais inferiores mas é conhecido por ter colisões[41] não foi utilizado. Assim optou-se por utilizar o SHA-1 em vez do SHA256 visto ambos terem as mesmas garantias mas SHA-1 ter valores de

tempos inferiores ao SHA256, ambos propostos pelo NIST[42]. Tempos esses que terão impacto significativo para grandes ficheiros.

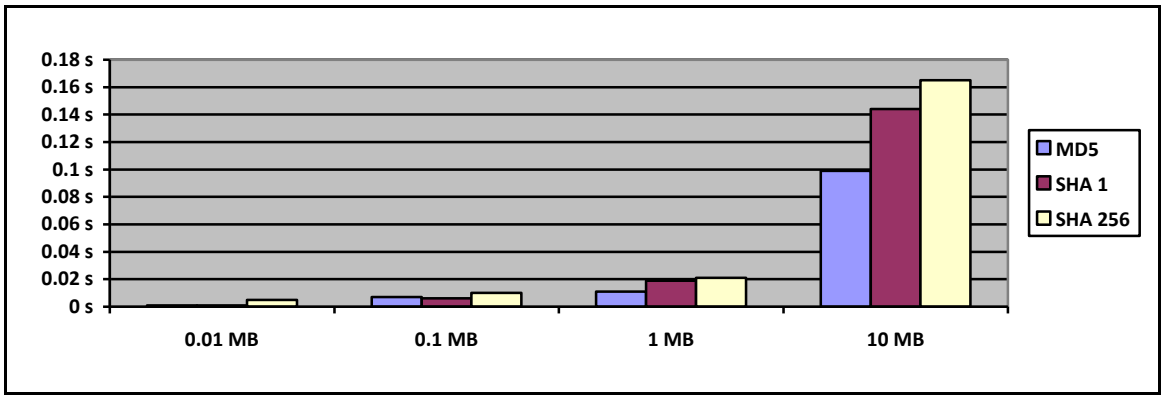

Figura 5.8: Gráfico correspondente aos valores da Tabela 5.9

5.3.3. Autenticação

Por fim o ultimo componente analisado deste middleware é o sistema de autenticação dos dados, sistema esse que permite como anteriormente apresentado garantir que um ficheiro foi criado por um utilizador do sistema previamente. Para este teste recorremos a dois algoritmos MD5withRSA e SHA1withRSA, recorrendo também a variação de tamanhos de ficheiros para os mesmos. Para este teste similarmente aos anteriores recorremos a uma média de 10 amostragens.

Tamanhos	MD5withRSA (seg)	SHA1withRSA (seg)
0.01 MB	0.009	0.011
0.1 MB	0.025	0.030
1 MB	0.062	0.073
10 MB	0.112	0.147

Tabela 5.10: Tempos (em segundos) de autenticação de ficheiros

Como é possível verificar na Tabela 5.10, os tempos de computação de códigos de autenticação são muito semelhantes. Optou-se por utilizar MD5withRSA, embora seja criptografia assimétrica o objetivo é garantir a autenticação visto que a integridade é tratada por outro componente. Assim o objetivo de autenticidade é facilmente satisfazível pela utilização do algoritmo MD5withRSA.

5.4. Vantagens do componente de profiling

Nesta secção avaliamos o middleware FairSky no aspeto do componente de *profiling*. Tanto em aspetos de redução de latências como em redução de custos de operação para o utilizador. Inicialmente colocamos o *profiler* em modo simples sem migrar os dados entre as clouds enquanto que em seguinte conjunto de testes comparamos os valores obtidos sem a migração face a migração de dados. Para este conjunto de testes nas medições de custos utilizamos tarifários reais de 4 provedores.

5.4.1. Minimização de latência sem migração de dados

5.4.1.1 Latências em leituras

Para este teste usamos, de modo semelhante aos anteriores, comparamos os resultados da operação do sistema FairSky com a escolha de leitura da cloud aleatória face a utilização do componente de *profiling*. Como abordado em capítulos anteriores, espera-se que o sistema de *profiler* quando configurado para reduzir as latências permita na maioria dos casos optar por utilizar a cloud mais rápida. O teste efetuado pressupõe que o ficheiro acedido encontra-se replicado por todas as clouds acima mencionadas sendo que assim a escolha é efetuada sobre todos os provedores.

Como podemos observar na Tabela 5.11, temos dois modos de operação, o modo sem *profiler* cujos resultados são baseados em escolha aleatória da cloud a usar, e o modo de operação com *profiler*. Como era de esperar o sistema de *profiling* permite uma amostragem mais solida e reduzida em comparação com a não utilização do componente. A falta do componente mostra valores cujo desvio padrão ronda os 2 segundos para a maioria das amostras. Isto deve-se ao facto que nas 10 amostras recolhidas algumas desses foram com base em acessos a clouds cujos tempos de leitura são mais elevados. Por outro lado a utilização de *profiler* permite redução significativa dos tempos de leitura, sendo que em todas as amostras recolhidas

Tamanho de Ficheiro	FairSky sem *Profiler* (seg)	FairSky com *Profiler* (seg)
0.01 MB	2.988 ±2.340	0.801 ±0.118
0.1 MB	3.398 ±2.403	1.183 ±0.231
1 MB	5.373 ±2.491	3.151 ±0.168
10 MB	24.742 ±2.725	20.512 ±0.587

Tabela 5.11: Tempos (em segundos) de operação de leitura

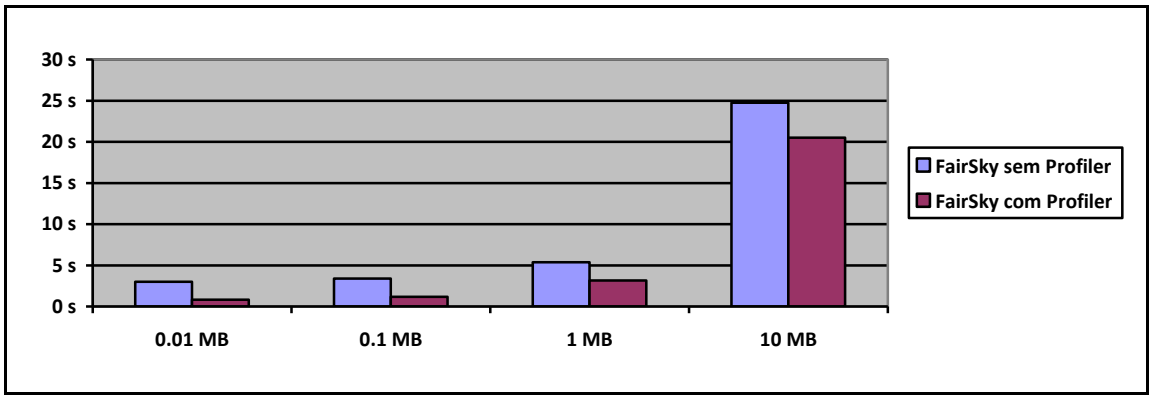

Figura 5.9: Gráfico correspondente aos valores da Tabela 5.11

5.4.1.2 Latências em escritas

Ao contrário do teste anterior, para este teste efetuamos uma escrita de um ficheiro nas clouds, sem que haja replicação do mesmo, pois essa replicação iria anular a utilidade do sistema de *profiling*. Para este teste espera-se permitir uma redução de tempos de escrita nas clouds.

A Tabela 5.12 apresenta os valores dos testes efetuados variando os tamanhos dos ficheiros para escritas do middleware sem o componente de *profiling* face aos tempos de escrita com o mesmo componente ativado. Como podemos observar os tempos recolhidos com a utilização do *profiler* são inferiores em todos os casos aos valores recolhidos nas medições sem o componente.

Nos testes sem o *profiler*, como a escrita é efetuada sobre uma cloud escolhida aleatoriamente sobre o conjunto de clouds disponibilizadas e como vimos inicialmente a comparação das mesmas, justifica-se o desvio padrão elevado como também a media de valores alta. Por outro lado, a utilização do *profiler*, como seria espectável, permite antes de cada escrita decidir com base em métricas recolhidas anteriormente indicar com alguma precisão qual a cloud mais rápida para a operação. Deste modo as médias são inferiores com a utilização do *profiler*. Note-se que para ficheiros grandes o sistema não obtém valores inferiores que a utilização da "melhor" clouds, nomeadamente a Dropbox, isto deve-se ao facto que para esse tamanho de ficheiros a Google Drive (como apresentado na secção 5.2.1) apresenta valores inferiores a Dropbox.

Tamanho de Ficheiro	FairSky sem Profiler (seg)	FairSky com Profiler (seg)
0.01 MB	3.988 ±2.754	1.295 ±0.289
0.1 MB	4.788 ±2.222	2.014 ±0.281
1 MB	14.150 ±1.379	12.161 ±0.559
10 MB	117.584 ±2.202	109.067 ±0.237

Tabela 5.12: Tempos (em segundos) de escrita de ficheiros

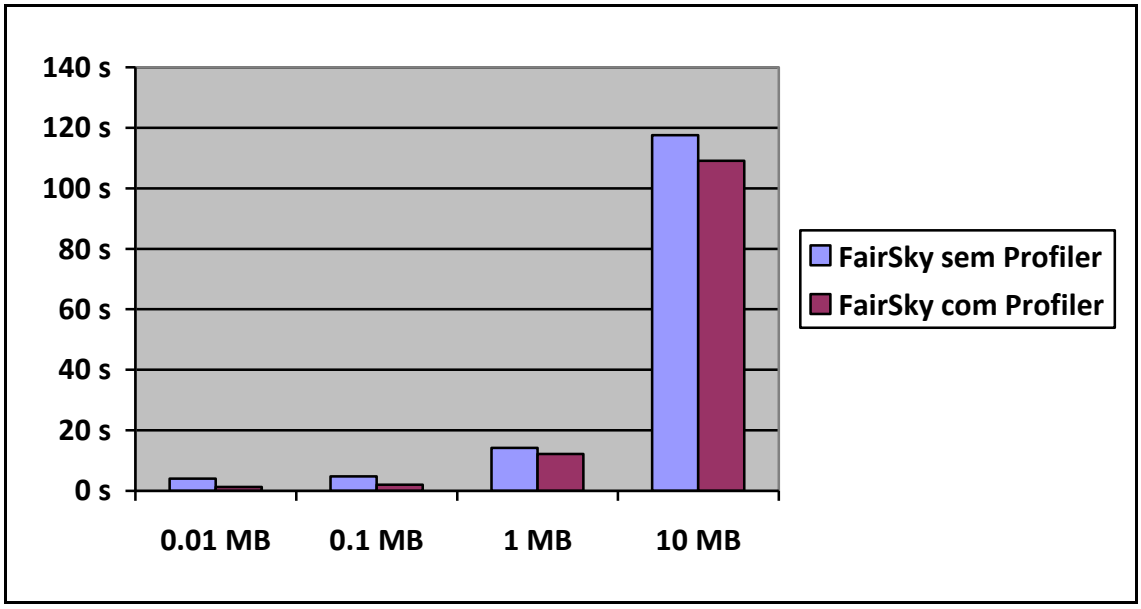

Figura 5.10: Gráfico correspondente aos valores da Tabela 5.12

5.4.2. Minimização de custos sem migração de dados

Para este teste recorremos ao outro conjunto de clouds apresentadas ao início. Clouds essas que oferecem tarifários pagos ao utilizador consoante o uso dos seus serviços taxando-os numa base mensal por volume e por tráfego. A referência para os custos disponíveis a partir dos diversos provedores envolvidos[21] está resumida na seguinte tabela de custos utilizada como referência para as avaliações que se indicam a seguir.

	Amazon S3	Microsoft Azure	Google Cloud	Rackspace
Armazenamento US $ / GB	0.095	0.070	0.085	0.100
Tráfego US $ / GB	0.000	0.000	0.120	0.120
PUT/GET/DELETE US $ / Operações	0.005 / 1000	0.0001 / 1000	0.01 / 1000	0.000

Tabela 5.13: Custos utilizados relativa às avaliações apresentadas a seguir

[21] https://aws.amazon.com/s3/pricing/ (acedido 13/12/13)
http://www.rackspace.com/cloud/files/pricing/ (acedido 13/12/13)
http://www.windowsazure.com/en-us/pricing/details/storage/ (acedido 13/12/13)
https://developers.google.com/storage/pricing (acedido 13/12/13)

Os custos foram calculados para 2 ficheiros de 50 GB, sendo um deles acedido uma vez por iteração do algoritmo, enquanto que o segundo não é acedido em momento algum sem ser a sua submissão no sistema. O objetivo deste tipo de acessos é criar um ficheiro que seja de uso frequente, e um ficheiro que nunca será acedido pelo sistema, sendo esse um ficheiro de arquivo. Assim cada iteração do sistema pode ser vista como um mês de faturação. Tendo assim um esquema de 1 acesso por iteração. Esperamos assim que o sistema FairSky apresente os custos da melhor cloud disponibilizada.

Na Tabela 5.14 podemos observar que dos 4 provedores pagos, a Microsoft Azure Storage contem o tarifário mais barato para o utilizador neste conjunto de testes. Assim como se pode verificar, o sistema FairSky tem idênticos custos que a Azure Storage, sendo que podemos concluir seguramente que essa foi a cloud escolhida pelo sistema. Este tipo de *profiling* permite que o utilizador delega a escolha da melhor cloud ao sistema em termos de custos.

Iterações	Amazon S3 US $	Microsoft Azure US $	Google Cloud US $	Rackspace US $	FairSky US $
10	142.04	70.02	129.04	166.05	70.02
20	278.09	140.04	252.08	326.11	140.04
30	414.14	210.07	375.12	486.16	210.07

Tabela 5.14: Custos ao utilizador por cada iteração

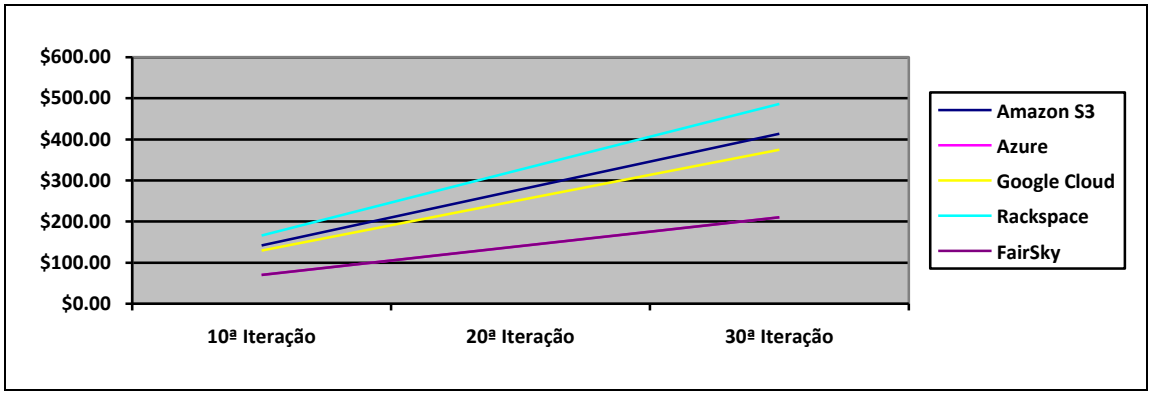

Figura 5.11: Gráfico correspondente aos valores da Tabela 5.14

5.4.3. Minimização de latências com migração de dados

Tendo efetuado anteriormente a avaliação do sistema face a utilização do *profiler* com redução de latências, vamos em seguida abordar a funcionalidade de migração. Para este teste avaliamos só as leituras de ficheiros nas clouds, visto que a operação de escrita não difere se houver ou não migração de dados, sendo que os dados escritos, como anteriormente visto, utilizam sempre

como repositório de escrita a melhor cloud. É espectável que as leituras sejam mais rápidas com a migração de dados.

Para efeitos deste teste começamos inicialmente com um subconjunto de provedores onde não se inclui a Dropbox sendo essa denominada por "melhor" cloud. Sendo que depois a mesma é inserida no sistema, observando agora a migração de dados com a nova cloud a usar, cloud essa que apresenta melhores tempos. Como se verifica o sistema FairSky apresenta uma redução de latências na operação de leitura, sendo que os dados agora encontram-se alojados na Dropbox. O facto de o sistema calcular a media de latências não permite que o mesmo delibere sobre qual cloud é melhor para cada tipo de ficheiros logo há um ganho nas latências para ficheiros onde a Dropbox é efetivamente mais rápida, nomeadamente ficheiros pequenos (10KB, 100KB e 1 MB) mas para ficheiros maiores há um aumento de latências pois os dados antes da adição estavam alojados na Google Drive que apresenta latências reduzidas para ficheiros grandes.

Em linhas gerais a migração de dados permite que a certa altura possamos adicionar um provedor e que os dados sejam migrados para ele caso ele apresente latências inferiores, por outro lado a escolha de extração de métricas baseada geralmente em médias de amostras de operações anteriores pode prejudicar certos tamanhos de ficheiros.

Tamanho de Ficheiro	FairSky antes adição (seg)	FairSky depois de adição (seg)
0.01 MB	1.567 ±0.152	0.815 ±0.197
0.1 MB	1.808 ±0.324	1.080 ±0.249
1 MB	3.251 ±0.226	3.141 ±0.235
10 MB	20.711 ±0.352	22.718 ±0.751

Tabela 5.15: Tempos (em segundos) de leitura de ficheiros

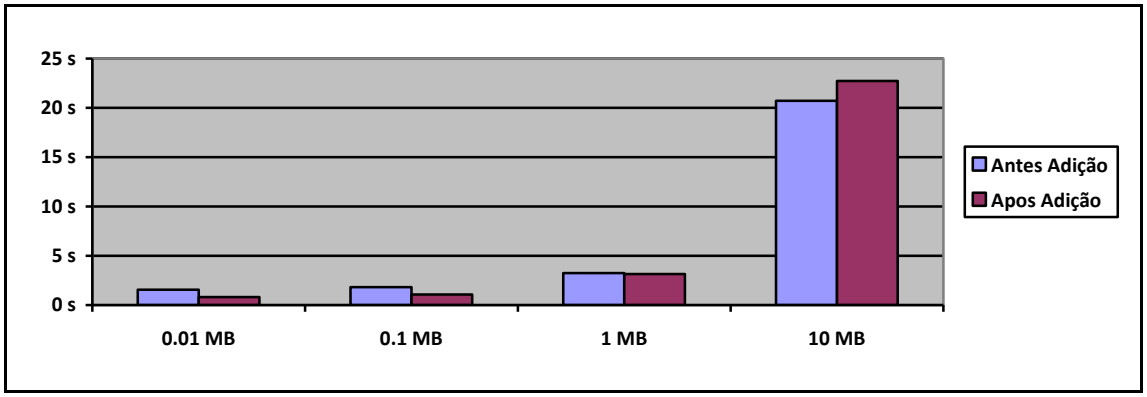

Figura 5.12: Gráfico correspondente aos valores da Tabela 5.15

5.4.4. Minimização de custos com migração de dados

Neste teste a semelhança do teste de custos anterior, recorremos ao outro conjunto de provedores que fornecem tarifários pagos. Usamos de mesmo modo dois ficheiros de 50GB cada aos quais efetuamos acesos constantes a um deles e ao outro não, deste modo destacamos que um ficheiro é de uso frequente e o outro para arquivar. Esperamos deste teste que o componente de migração seja vantajoso pois ele devera migrar objetos com pouco uso para provedores mais adequados a esse tipo de frequência de acessos permitindo uma redução de custos ao utilizador ao final de cada iteração.

Como podemos observar na Tabela 5.16, ao final de 30 iterações do sistema houve uma redução de custos de utilização de 3.06$ face ao mesmo esquema de utilização mas sem o componente de migração ativo. Na 10ª iteração observa-se um valor superior na versão com migração em comparação com a equivalente sem migração. Isto deve-se ao facto do sistema decidir que o segundo ficheiro devera ser arquivado num provedor mais barato. Provedor esse que é classificado pelo custo de armazenamento e não pelo custo de tráfego de dados. Na 20ª iteração reparamos que o custo pago na 10ª iteração começa a ser rebatido sendo ambas as vertentes com custos semelhantes, enquanto que, na 30ª iteração a vertente de migração apresenta uma redução de 3.06$ para o utilizador, redução essa alcançada migrando o segundo ficheiro para outro provedor.

Concluímos com este estudo que o componente de migração poderá em certas alturas ser mais caro ao utilizador devido aos custos de migração de dados entre as clouds, mas esse custo mais tarde será convertido em redução do mesmo.

Iterações	FairSky sem Migração US $	FairSky com Migração US $
10	70.02	73.93
20	140.04	140.47
30	210.07	207.01

Tabela 5.16: Custos de utilização do sistema

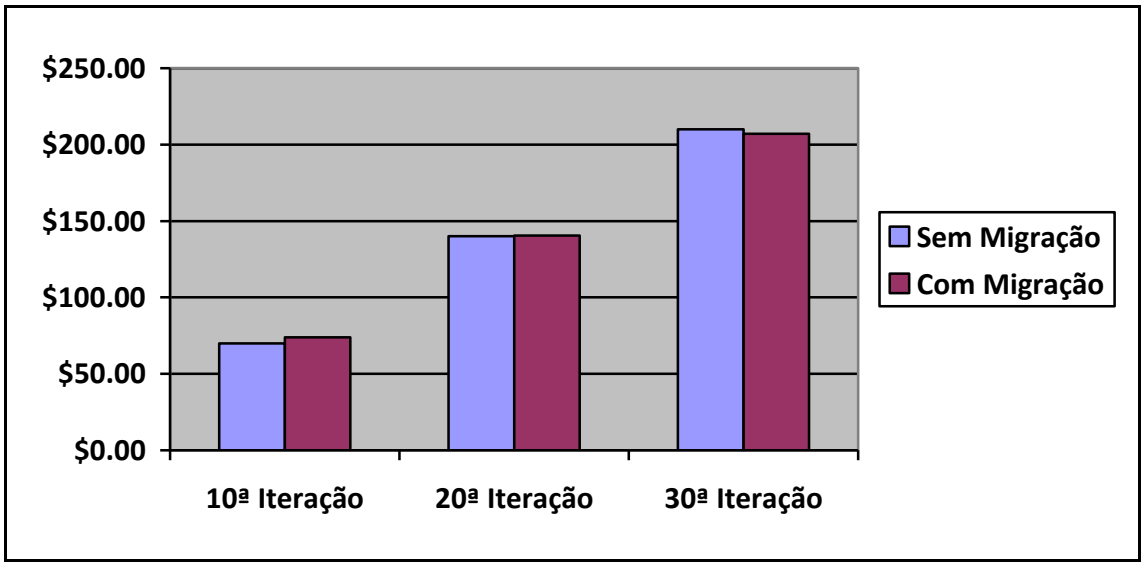

Figura 5.13: Gráfico correspondente aos valores da Tabela 5.16

5.5. Discussão

Neste capítulo foi feita uma avaliação do sistema desenvolvido, face a utilização do *profiler* recorrendo a provedores de armazenamento publico tal como são disponibilizados atualmente. Foram usadas quadro provedores públicos de armazenamento, todos eles com tarifários gratuitos ate certo limite de dados e sem opção de escolha de localização dos dados do utilizador. Fator que é da inteira responsabilidade do provedor.

Inicialmente foram analisados os diversos provedores de clouds em questões de latências, sendo desse teste elegido a Dropbox como a cloud cuja latência na maioria dos casos foi a menor em comparação com as restantes. Em seguida apresentou-se o funcionamento do middleware FairSky utilizado sobre a Dropbox face a utilização da mesma diretamente. Observou-se que o impacto do middleware é muito pequeno face aos tempos de transferência de dados observados como também o acréscimo de segurança e integridade oferecido pelo mesmo, algo que a Dropbox não oferece.

Tendo demonstrado que o acréscimo de tempo face ao ganho é justificável, abordamos a comparação do uso de uma cloud no FairSky face a utilização de quatro em simultâneo sendo que foi debatida a utilização de replicação e o seu impacto no sistema. Como era de esperar a utilização de um provedor aleatório para leitura torna com que o sistema fique no pior caso dependente dos tempos e desempenho dessas clouds, algo que os valores, especialmente os desvios padrões apresentados demonstraram. Por outro lado a necessidade de replicar os dados pelas 4 clouds obriga que haja interação com todas sendo que nesse caso estamos sempre pendentes da pior cloud disponível. Como as operações dos diversos componentes operam de forma independentemente do número de clouds a usar, nomeadamente os componentes processam os fi-

cheiros uma vez antes de os submeter nas clouds ou só após de ter os dados escritos nelas, no caso de uma leitura, os seus tempos são sempre variados em função do tamanho de dados. Assim em seguida apresentou-se o impacto dos mesmos variando as configurações consideradas como também a diferenciação entre tamanhos de ficheiros. A confidencialidade é garantida, recorrendo a algoritmos criptográficos, nomeadamente AES 256. Foi escolhido este devido ao ganho de propriedades de segurança face ao tempo acréscimo necessário do mesmo em comparação com as alternativas propostas de AES 128, AES 192 e Triple DES. O Triple DES foi rejeitado devido a grande diferença de tempo de processamento necessário face aos algoritmos AES.

Para a integridade de dados foram testados três algoritmos, MD5, SHA1 e SHA-256, dos quais se utilizou SHA1. O algoritmo SHA1 oferece garantias de integridade como também tolerância a colisões, garantias essas oferecidas pela variante SHA-256. Mas com o aumento do tamanho dos ficheiros o algoritmo SHA-256 começa a ocupar bastante tempo de processamento aumentando assim os tempos globais de escritas e leituras de ficheiros nas diversas clouds. O algoritmo MD5 foi rejeitado devido ao facto de haverem colisões no mesmo podendo assim a um atacante alterar o ficheiro na cloud sem que se comprometa a integridade do mesmo.

Por fim, na avaliação de componentes aborda-se o componente de autenticidade. Para este utilizou-se MD5withRSA face a alternativa analisada de SHA1withRSA. Isto deve-se ao tempo de processamento inferior por parte do MD5withRSA face a alternativa sendo que a integridade é tratada pelo módulo de integridade, e o objetivo é perante uma chave privada do utilizador produzir o código de autenticidade do conteúdo. Assim não há preocupação de criação de colisões ou forjar o conteúdo passando-se por um utilizador legítimo sem que seja detetado pelo mecanismo de autenticidade. Importante é salientar que a criptografia assimétrica é muito mais lenta que a criptografia assimétrica assim, escolhe-se a alternativa com menor custo para o sistema.

Por fim abordou-se a utilização do sistema de *profiling* tanto em questões de medições de latência como medições de acessos frequentes. Em primeiro plane estudou-se o uso unicamente do mesmo para escritas e leituras de ficheiros onde demonstrou redução de latências optando sempre por escolher a cloud mais rápida. Importante será salientar que as métricas afetam um todo, logo as otimizações não tem em consideração os desempenhos diferentes das clouds para tamanhos diferentes algo que poderá ser problemático se temos uma grande variedade de tamanhos, muito pequenos e muito grandes. Por outro na questão de custos permitiu-se simplesmente que o sistema utilizasse unicamente a cloud mais vantajosa de um conjunto delas.

A migração de dados na questão de redução de latências e de custos provou ser um aspeto interessante no sistema. Em contrário com as leituras de ficheiros onde para otimizar as leituras

devem-se replicar os dados por todas, com a migração basta que um ficheiro seja colocado uma vez nas clouds, reduzindo o espaço necessário nelas, sendo que para reduzir os custos o middleware periodicamente migra os dados de um provedor para outro constante os tempos de latências. Para este teste optou-se por adicionar a Dropbox a meio da amostragem sendo que era esperável que os dados a uma certa altura depois da adição migrassem para a nova. Por outro o componente de redução de custos permitiu reduzir ainda mais os custos sendo que os dados que foram classificados de pouco acesso foram colocados em clouds mais baratas para esse tipo de acessos.

Assim concluímos que apesar de certos aspetos não serem levados em consideração, nomeadamente o tamanho de ficheiros em função de migração ou leitura da melhor cloud, a utilização de um mecanismo de *profiling* demonstra ser um componente útil e interessante para redução de latências e de custos para o utilizador.

6

Conclusão e Trabalho Futuro

O trabalho realizado nesta dissertação teve como objetivo modelar e desenvolver um sistema middleware responsável pela gestão confiável e segura de dados sensíveis armazenados na cloud, numa solução de cloud de clouds, tendo também como prioridade a otimização da utilização das mesmas, quem em termos de latências ou em termos de custos para o utilizador. Estes dados correspondem a ficheiros genéricos, colocados pelo utilizador, que detém todo o controlo da base de confiança do sistema, que intermedeia uma aplicação de um utilizador e as várias clouds heterogéneas não necessariamente seguras ou confiáveis, utilizadas como *outsourcing* para o armazenamento dos dados.

Como base na análise de diversos sistemas que tinham como objetivo o endereçamento desta problemática, sendo que nenhum abordava por completo a combinação de redução de latências ou custo com confiabilidade e segurança de dados, foi proposto e implementado um sistema *middleware* genérico, com uma API semelhante aos serviços de armazenamento de dados na *cloud* existentes hoje em dia. Esta API permite que as aplicações desenvolvidas em Java possam facilmente integrar o sistema FairSky, como repositório seguro de dados, como também oferecer as mesmas capacidades de redução de latências de operações sobre os repositórios finais como também reduzir os custos de utilização.

A solução desenvolvida deveria ter modularidade suficiente de forma a não constituir um único ponto de falha, bem como a possibilidade de executar a mesma em paralelo com outras instâncias sendo que possa assim permitir múltiplas aplicações a acederem a partir de diversos pontos sobre os mesmos dados. Assim foi desenvolvido o sistema FairSky, uma cloud de controlo de intermediação e acesso transparente de múltiplas *clouds* de dados heterogéneas onde os dados encontram-se fragmentados, cifrados, e distribuídos segundo os critérios configurados no

profiling a propósito de reduzir latências e custos de utilização recorrendo a migração de dados entre diversos provedores caso isso seja favorável ao utilizador, como também permitir que os dados sejam replicados seguindo um modelo de replicação bizantina.

O sistema *middleware* proposto, implementado e testado pode ser concretizado e operacionalizado com flexibilidade, em diferentes tipos de solução:

- Como sistema *middleware* executado localmente num computador de um utilizador, para suportar um serviço local de intermediação com as múltiplas clouds de armazenamento utilizadas.
- Como serviço "*proxy*", por exemplo numa rede local de uma empresa, permitindo ser usado como repositório virtual entre os utilizadores, através de aplicações executadas nesses mesmos computadores, sendo eles clientes deste serviço.
- Como uma cloud de servidores confiáveis (ou seja uma implementação baseada em serviços cloud), como uma solução escalável e condições de acesso ubíquo.

Na sua arquitetura de *software*, o FairSky disponibiliza um sistema confiável de acesso as múltiplas *clouds* de armazenamento, oferecendo uma API para leituras, escritas, remoções e de conteúdos, com o modelo de concorrência de semântica "*one writer – multiple readers*", garantindo deste modo propriedades de confiabilidade, segurança e privacidade.

O sistema de *profiling*, funciona periodicamente monitorizando os acessos aos ficheiros do sistema, sendo com essa informação extraída permite decidir se um ficheiro devera ser migrado para outro provedor *cloud* cujo tarifário e prioridades estipuladas pelo utilizador permite reduzir os impactos do uso da implementação, sejam esses impactos financeiros ou de latência de operações efetuadas. Esses critérios e condições são sempre deixados a conta do utilizador o qual pode especificar a sua vontade as condições e situações onde um ficheiro deve ser migrado.

6.1. Objetivos

Tendo em conta os objetivos inicialmente propostos, podemos considerar que estes foram alcançados. Na implementação, teste e avaliação do sistema, enquanto arquitetura do tipo *middleware*, utilizável como cliente local, verificou-se um custo de processamento acrescido associado ao uso da solução aceitável, tendo em conta as propriedades de confiabilidade e segurança oferecidas face a não utilização do mesmo.

No que diz respeito à avaliação de desempenho do sistema, as contribuições desenvolvidas estão em conformidade com os requisitos especificados e fundamentados da solução proposta revelando ser interessante. Verificou-se ate que as operações são processadas em tempos muito idênticos, a exceção da utilização aleatória das clouds. Nesta questão a utilização do mó-

dulo de otimização de latências permitiu observar temos inferiores em relação a utilização para piores clouds, sendo essas usadas diretamente como repositório de dados.

A solução proposta apresenta propriedades de redução de custos e otimizações de latência para operações de escrita e leitura, sendo que para esse efeito o componente de *profiling* opera periodicamente sobre os dados extraindo métricas associadas a frequência de acessos a ficheiros, e recolhendo métricas de latência relativas a operações efetuadas anteriormente. Métricas essas sobre as quais efetua-se uma media para decidir qual a cloud mais rápida a utilizar. Ao utilizar este mecanismo verificou-se uma redução de custos de utilização dos provedores. Por outro lado a migração de dados permite ao utilizador colocar os dados e para além dos mesmos migrarem se o tarifário permitir maiores ganhos, reduzir latências sendo que a adição dinâmica de provedores poderá levar a utilização de uma *cloud* cujas latências são inferiores as restantes. Um aspeto interessante abordado na componente da migração é que permite otimizar latências sem que os dados estejam replicados por todas as clouds, algo que seria necessário se quiséssemos optar sempre pela melhor *cloud* mas era impedi tório devido aos tempos de escrita necessários para replicar um ficheiro.

A confidencialidade, autenticidade e integridade dos dados são salvaguardadas por processos criptográficos credíveis, sendo esse suporte transparente em relação à possibilidade de utilizar diferentes algoritmos criptográficos incorporados ou a incorporar na solução.

O suporte de múltiplas clouds heterogéneas é garantido pela utilização de diferentes conectores que permitem acesso a diferentes soluções heterogéneas de provedores na Internet, adaptando-se assim a particularidade de interface de cada um como também aos aspetos de processamento de dados, fornecendo assim internamente ao sistema uma camada de abstração tornando mais simples aos restantes módulos operarem sobre os provedores em si.

Por fim, a fiabilidade, tolerância a falhas bizantinas e disponibilidade permanente são garantidas para um quórum onde a equação $3f + 1$ onde o sistema replica os dados por essas clouds, e embora recorrendo ao sistema de *profiling*, consegue-se sempre reconstruir um ficheiro. Sendo que por exemplo para 4 *clouds* consegue-se garantir a integridade do mesmo a partir das restantes 3 *clouds*.

6.2. Trabalho Futuro

No que diz respeito ao trabalho futuro, existem diversos pontos que podem ser endereçados de forma a estender e melhorar o sistema implementado. Esses pontos focam-se nas três camadas do middleware, sendo eles:

- Suporte para um maior número de clouds de armazenamento de dados, com o desenvolvimento de conectores específicos para cada uma delas. Permite dotar o sis-

tema de maior resiliência face a falhas de mais que uma clouds seguindo o modelo $3f+1$. Permite também expandir as capacidades de otimização de custos sendo que existem muitos provedores de armazenamento pago na Internet que não foram endereçados aqui cujos tarifários podem possivelmente apresentar maior ganhos para o utilizador recorrendo ao componente de Profiler, como também provedores cujas latências sejam inferiores, nomeadamente utilizar provedores nacionais.

- Melhorar o sistema de suporte e concorrência para múltiplos utilizadores, sendo que o sistema atual não foi testado para essa funcionalidade embora implementada.
- Permitir autenticação e gestão de utilizadores sendo que possa-se permitir uma área de dados pessoais para cada utilizador registado no sistema.
- Validar extensamente o uso de um índice externo ao sistema, nomeadamente recorrer ao sistema de replicação de dados oferecido pelo OpenReplica.
- Seria interessante desenvolver uma aplicação com interface gráfica, que usaria a API através da qual o cliente poderá gerir os seus ficheiros. Nomeadamente uma aplicação do género FileZilla[22] ou sincronização de uma pasta local com o repositório, similarmente as aplicações oferecidas por diversos provedores, nomeadamente Dropbox.
- Permitir fragmentação de diferentes tamanhos no sistema para diferentes ficheiros sendo que presentemente encontra-se disponibilizada fragmentação de tamanho único para todos os ficheiros utilizados no sistema. Este tipo de fragmentação poderá permitir otimizar as latências de operações visto que certos provedores comportam-se melhor para tamanhos de ficheiros diferentes.
- Implementar um sistema de pesquisas de conteúdos ou de nomes de ficheiros permitindo facilmente indexar os dados e efetuar pesquisas sobre os mesmos.
- O sistema de *caching* poderá ser otimizado sendo que para além de ser ponto de acesso para otimização de acessos ao índice mas também otimização e minimização de custos relativos a acessos frequentes a ficheiros na clouds. Sendo que deste modo ao aceder a ficheiros na Cache permite-se uma redução tanto de custos de operação como de latências de acessos aos dados armazenados.
- Efetuar mais testes sobre ficheiros diferentes e com outro tipo de acessos sobre os dados sendo que poderá haver outros cenários de utilização que apresentem valores de custos mais favoráveis. Como também explorar o aspeto de reconfiguração di-

[22] FileZilla Project: https://filezilla-project.org/ (acedido a 17-11-2013)

nâmica sendo essa dos componentes ou da modelação dos critérios de migração de dados.

- Avaliar e otimizar o índice interno do sistema de ficheiros. Sendo que será interessante comparar o índice interno a outras ferramentas de indexação como por exemplo a comparação do índice interno face ao OpenReplica.

7

Bibliografia

[1] Amazon.com Inc., “Amazon S3.” [Online]. Available: http://aws.amazon.com/s3. [Accessed: 13-Dec-2013].

[2] Google Inc., “Google Drive.” [Online]. Available: http://drive.google.com. [Accessed: 13-Dec-2013].

[3] Microsoft Corporation, “Microsoft Windows Azure.” [Online]. Available: http://www.windowsazure.com/en-us/. [Accessed: 13-Dec-2013].

[4] Portugal Telecom SGPS SA., “Meo Cloud.” [Online]. Available: https://meocloud.pt. [Accessed: 13-Dec-2013].

[5] Rackspace US Inc., “Rackspace.” [Online]. Available: http://www.rackspace.co.uk/cloud-files/. [Accessed: 13-Dec-2013].

[6] Box Inc., “Box.” [Online]. Available: http://www.box.com. [Accessed: 13-Dec-2013].

[7] Google Inc., “Google Cloud Storage.” [Online]. Available: https://cloud.google.com/products/cloud-storage. [Accessed: 13-Dec-2013].

[8] Dropbox Inc., “Dropbox.” [Online]. Available: https://www.dropbox.com. [Accessed: 13-Dec-2013].

[9] Microsoft Corporation, “Microsoft SkyDrive.” [Online]. Available: http://skydrive.com/. [Accessed: 13-Dec-2013].

[10] Apple Inc., “iCloud.” [Online]. Available: https://www.icloud.com/. [Accessed: 13-Dec-2013].

[11] Canonical Ltd., “Ubuntu One.” [Online]. Available: http://one.ubuntu.com/. [Accessed: 13-Dec-2013].

[12] Hasimed, “Hasimed Cloud Service Provider.” [Online]. Available: http://www.hasimed.com. [Accessed: 13-Dec-2013].

[13] Xignite, “Market Data Cloud.” [Online]. Available: http://www.xignite.com/market-data/market-data-cloud/. [Accessed: 13-Dec-2013].

[14] Z. Wu, M. Butkiewicz, and D. Perkins, “SPANStore: cost-effective geo-replicated storage spanning multiple cloud services,” in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, 2013.

[15] H. Abu-Libdeh, “RACS: a case for cloud storage diversity,” in *Proceedings of the 1st ACM symposium on Cloud computing*, 2010.

[16] R. Kotla, L. Alvisi, and M. Dahlin, “SafeStore: a durable and practical storage system,” in *2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*, pp. 129–142, 2007.

[17] G. A. Alvarez, E. Borowsky, S. Go, T. H. Romer, R. Becker-szendy, R. Golding, A. Merchant, M. Spasojevic, A. Veitch, and J. Wilkes, “MINERVA : An Automated Resource Provisioning Tool for Large-Scale Storage Systems,” in *ACM Transactions on Computer Systems*, vol. 19, no. 4, pp. 483–518, 2001.

[18] E. Anderson, M. Hobbs, K. Keeton, and S. Spence, “Hippodrome: Running Circles Around Storage Administration.,” in *Proceedings of the 1st USENIX Conference on File and Storage Technologies*, no. January, pp. 175–188, 2002.

[19] J. Wilkes, “Traveling to Rome: QoS specifications for automated storage system management,” in *Proceedings of the 9th International Workshop on —IWQoS*, no. June, 2001.

[20] K. A. Smith, “File system aging---increasing the relevance of file system benchmarks,” in *ACM SIGMETRICS Performance Evaluation Review*, vol. 25, no. 1, pp. 203–213, 1997.

[21] R. Jammalamadaka and R. Gamboni, “iDataGuard: an interoperable security middleware for untrusted internet data storage,” in *Proceedings of the ACM/IFIP/USENIX Middleware ’08 Conference Companion*, pp. 36–41, 2008.

[22] K. Puttaswamy, C. Kruegel, and B. Zhao, “Silverline: toward data confidentiality in storage-intensive cloud applications,” in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, 2011.

[23] K. Bowers, A. Juels, and A. Oprea, “HAIL: a high-availability and integrity layer for cloud storage,” *Proceedings of the 16th ACM conference on Computer and Communications Security*, vol. 489, pp. 187–198, 2009.

[24] A. Bessani, M. Correia, and B. Quaresma, “DepSky: dependable and secure storage in a cloud-of-clouds,” in *EuroSys’11 - Architecture*, pp. 31–45, 2011.

[25] C. Smith and E. Miller, “An Improved Long-Term File-Usage Prediction Algorithm,” in *25th Annual International Conferenceon Computer Measurement and Performance*, 1999.

[26] Gartner, “Gartner Executive Programs Worldwide Survey of More Than 2,000 CIOs Identifies Cloud Computing as Top Technology Priority for CIOs in 2011,” *Gartner*, 2011. [Online]. Available: http://www.gartner.com/newsroom/id/1526414. [Accessed: 24-Jan-2013].

[27] Cloud Security Alliance, “Top Threats to Cloud Computing,” in *Security*, no. March, pp. 1–14, 2010.

[28] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, “Hey, you, get off of my cloud,” in *Proceedings of the 16th ACM conference on Computer and communications security CCS*, vol. 45, p. 199, 2009.

[29] M. Jensen, J. Schwenk, J.-M. Bohli, N. Gruschka, and L. Lo Iacono, “Security Prospects through Cloud Computing by Adopting Multiple Clouds,” in *Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing*, pp. 565–572, 2011.

[30] “Liquid Motors, Inc. v. Allyn Lynd and United States of America. U.S. District Court for the Northern District of Texas, Dallas Division.” 2009.

[31] R. Chow, P. Golle, and M. Jakobsson, “Controlling data in the cloud: outsourcing computation without outsourcing control,” in *Proceedings of the 2009 ACM workshop on Cloud computing security*, pp. 85–90, 2009.

[32] R. C. Jammalamadaka, R. Gamboni, S. Mehrotra, and K. E. Seamons, “iDataGuard : Middleware Providing a Secure Network Drive Interface to Untrusted Internet Data Storage Adopting to Heterogeneity,” pp. 710–714.

[33] L. Lamport, R. Shostak, and M. Pease, “The Byzantine Generals Problem,” in *ACM Transactions on Programming Languages and Systems*, vol. 4, no. 3, pp. 382–401, 1982.

[34] M. Castro and B. Liskov, “Practical Byzantine Fault Tolerance,” in *ACM Transactions on Computer Systems*, vol. 20, no. 4, pp. 173–186, 1999.

[35] M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. J. Wylie, “Fault-scalable Byzantine fault-tolerant services,” in *ACM SIGOPS Operating Systems Review*, vol. 39, no. 5, pp. 59–74, 2005.

[36] J. Cowling, D. Myers, B. Liskov, R. Rodrigues, and L. Shrira, “HQ Replication : A Hybrid Quorum Protocol for Byzantine Fault Tolerance,” in *Symposium A Quarterly Journal In Modern Foreign Literatures*, pp. 177–190, 2006.

[37] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, “Zyzzyva : Speculative Byzantine Fault Tolerance,” in *Transactions on Computer Systems (TOCS)*, vol. 41, no. 6, pp. 45–58, 2007.

[38] A. Clement, M. Kapritsos, S. Lee, Y. Wang, L. Alvisi, M. Dahlin, and T. Riche, "Upright cluster services," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles SOSP*, vol. 9, p. 277, 2009.

[39] L. Gillam, B. Li, J. O'Loughlin, and A. S. Tomar, "Fair Benchmarking for Cloud Computing Systems," in *Journal of Cloud Computing: Advances, Systems and Applications*, 2013.

[40] M. G. Baker, J. H. Hartman, M. D. Kupfer, K. W. Shirriff, and J. K. Ousterhout, "Measurements of a distributed file system," in *ACM SIGOPS Operating Systems Review*, vol. 25, no. 5, pp. 198–212, 1991.

[41] X. Wang and H. Yu, "How to break MD5 and other hash functions," in *Advances in Cryptology–EUROCRYPT*, 2005.

[42] P. Gallagher, "Secure Hash Standard (SHS)," in *Federal Information Processing Standards Publication 180-3*, no. March, 2008.